Formal Specifications Better Than Function Points for Code Sizing

Mark Staples, Rafal Kolanski, Gerwin Klein, Corey Lewis, June Andronick, Toby Murray, Ross Jeffery, Len Bass
NICTA, 13 Garden St, Eveleigh NSW 2015, Australia
School of Computer Science and Engineering, University of New South Wales 2052, Australia
email: firstname.lastname@nicta.com.au

***Abstract*—Size and effort estimation is a significant challenge for the management of large-scale formal verification projects. We report on an initial study of relationships between the sizes of artefacts from the development of seL4, a formally-verified embedded systems microkernel. For each API function we first determined its COSMIC Function Point (CFP) count (based on the seL4 user manual), then sliced the formal specifications and source code, and performed a normalised line count on these artefact slices. We found strong and significant relationships between the sizes of the artefact slices, but no significant relationships between them and the CFP counts. Our finding that CFP is poorly correlated with lines of code is based on just one system, but is largely consistent with prior literature. We find CFP is also poorly correlated with the size of formal specifications. Nonetheless, lines of formal specification correlate with lines of source code, and this may provide a basis for size prediction in future formal verification projects. In future work we will investigate proof sizing.**

I. Introduction

For development projects, size and effort estimates can determine available resources, and are a criterion for judging their success. So accurate estimation is critical. We are developing approaches to enable the cost-effective development of dependable embedded systems, backed by code-level formal verification of an embedded system microkernel [1] and key trusted components. However, as discussed in a prior paper [2], estimation for formal methods projects is hard [3]. They create and use new kinds of artefacts, follow different process lifecycles, use different development and verification technologies. It is not known how to calibrate measures of formal methods work to traditional metrics approaches [4], and this is exacerbated by limited data on formal methods projects [5]. We are investigating software sizing methods to develop validated artefact size models for embedded systems formal verification projects. We plan to use these sizing models to create effort estimation models for such projects.

In this paper, we report on an initial study of relationships between size measurements of artefacts from the formal verification of the seL4 embedded systems microkernel. We compare normalised line counts of specification and code artefacts in the project with each other and with COSMIC Function Points (CFP) counts. CFP is a simple measure of software functional size designed to be broadly applicable, including for embedded and real-time software [6]. We find poor correlations between CFP counts and the size of both code and formal specifications. However, we do find that the size of formal specifications have a strong and significant relationship to the size of the C source code.

II. Background

A. seL4 and Large-Scale Formal Verification

The L4.verified project completed the code-level formal verification of the full functional correctness of the seL4 embedded systems microkernel [1]. seL4 is a security-enhanced microkernel in the L4 family [7]. We previously described [2] a middle-out lifecycle model and timeline for this project. The first artefact to be developed was an *executable specification*. This was written in Haskell, and served as a design prototype. It was later automatically transliterated to higher-order logic in the Isabelle/HOL theorem prover [8] for use as a design-level formal specification. The next artefact was the *abstract specification*, written directly in Isabelle/HOL. The executable specification describes the algorithms and data structures used, whereas the abstract specification describes higher-level behaviour without reference to any specific low-level algorithms or data-structures. Finally, the *source code*, in C, was developed and captured in an operational semantics for C represented in Isabelle/HOL. Isabelle/HOL was used to check proofs of correctness of the C source code with respect to the abstract specification (via the executable specification).

This project was relatively large, taking around 25 person-years to complete the formal specification, development, and formal verification of seL4 [1]. The size and complexity of this effort has revealed challenges in managing large-scale formal verification projects. A basic challenge is how to size artefacts in such projects [2], to inform cost and effort estimation, and to validate and calibrate process simulation models [9].

B. Software Size Measures

Measuring the size of software is a long-standing challenge in software engineering. The most widespread measure remains line counts, usually normalised by excluding comments and whitespace. However, source code is often only available late in the project lifecycle, and so is not a good target for early lifecycle estimation.

Other artefacts, such as requirements and designs, can also be used to size software. The COSMIC Function Point (CFP) method [10] is an internationally standardised way of sizing software based on its requirements. The method has been successfully applied in a number of industrial settings [11]–[13]. The method typically starts with a *Functional User Requirements* document, from which a number of event-triggered *functional processes* are identified. The functionality of these is represented by a sequence of data movement operations (*sub-processes*): Entry, Exit, Read, or Write operations. The CFP count is the number of these operations.

III. Method

Our primary goal was to investigate whether CFP is a good candidate as a predictor for the size of the major artefacts in the L4.verified project. We examined the size of the artefacts at three different levels: the abstract specification, an executable specification, and the source code.

For our unit of analysis, we used the functions in the microkernel API as documented in the seL4 user manual [14]. The manual describes a conceptual model for seL4 and documents the API functions. We identified 31 API functions: 25 architecture-independent functions, and 8 architecture-dependent functions. We considered only the functions in the verified ARM version of seL4. Below we detail our method for sizing the project artefacts and deriving the CFP counts.

A. Sizing Project Artefacts with Line Counts

To measure the size of artefacts we use line counts normalised by excluding white space and comments. The layout of files is not fully normalised, such as with the use of a pretty-printer, but there is a largely consistent format, and prior research has indicated that formatting differences do not usually cause significant variation in line counts [15]. For C, line counting occurred after macro expansion and was performed by the same C parser used in the L4.verified project. A locally-developed Isabelle specification line counting program was used for the abstract and executable specifications.

To measure the size of each API function in each artefact, we computed the set of internal functions that are (transitively) called for that function, termed the API *slice*, and then took the sum of the normalised line count of each function in the slice. We first manually determined the top-level functions for each API call, and then automatically computed the slice by taking the transitive closure over the callgraph of the artefact. The manual step was performed by different researchers for each of the artefact types, who later cross-checked each other's results for consistency. Because each API function was sliced independently, shared sub-functions are double-counted across functions. This arguably matches the measure of functionality captured by CFP counts.

B. Sizing APIs with COSMIC Function Point Counts

CFP counts are notionally performed on a *Functional User Requirements* (FUR) document, which can be made available near the beginning of a project. However, the CFP standard [6] recognises that CFP counts sometimes have to rely on other sources: the important thing to measure is functionality delivered to users. There is no FUR for seL4, and so instead we used the seL4 user manual [14], which has a similar target audience (users) and level of abstraction as a FUR. This manual was produced towards the end of the L4.verified project, but could in principle have been documented very early in the project timeline.

The seL4 API functions are a reasonable target for CFP counting. Each function can be seen as a CFP *functional process* triggered by external events, with a set of inputs, internal behaviours (involving reads and writes), and outputs.

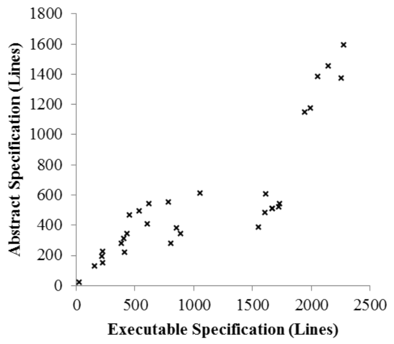
According to the CHAR method [16], seL4 is a *Data Driven Control System.* Its functionality is dominated by control and communication functionality, but it also has some level of data complexity functionality associated with its capability-based access control mechanisms. There is no scientific or adaptive functionality directly within seL4, and the data manipulation functionality is related more to the complexity of internal data structures rather than complex processing of that data. CFP is intended to be applicable to real-time systems [6] and has been successfully applied to embedded systems in this functional domain [17], [18].

As recommended by prior authors [19], we ran initial pilot counting sessions to agree on a consistent CFP counting methodology, including agreement on suitable level of abstraction and objects of interest. The objects and level of abstraction were those described in the user manual. We recorded the functionality of each API function as scripts of CFP operations. In our CFP scripts, we distinguished special operations. We identified *Resolve* operations that performed two levels of dereferencing of capability references to microkernel-protected kernel objects, which we took as sugar for two *Read* operations. We also identified *Create* and *Delete* operations which we took as sugar for *Write* operations. For consistency, one researcher documented all of these CFP scripts, and for reliability, the scripts were independently reviewed. The operations were cross-checked with a UML class diagram that defined entities and data attributes in a conceptual model for seL4. The CFP counts were calculated from the scripts.

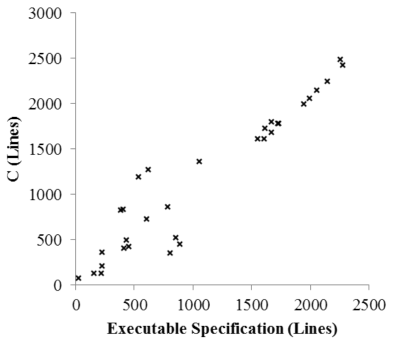
IV. Results

Figure 1 shows the scatterplot relationships between the size (in lines) of the major artefacts. All are statistically significant at the 0.05 level, and all have strong correlations near 0.9. Figure 2 shows similar scatterplots for the relationship between CFP and each of the major artefacts. We used version 20 of SPSS to analyse the data. None of the CFP relationships are statistically significant at the 0.05 level, and have much weaker correlations in any event. The CFP and line counts all satisfy the Kolmogrov-Smirnoff test of normality, except for the abstract specification line counts ($p = 0.025$). So we have reported Pearson's correlation coefficient, except for relationships involving the abstract specification for which we report Spearman's rho.

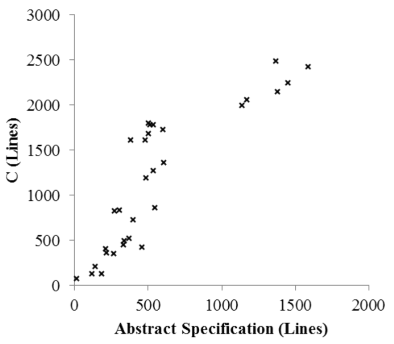
The abstract specification line counts appear bimodal, and one might imagine that the CFP versus line count scatterplots have two clusters, separated at line counts for executable specification > 1500, abstract specification > 1000, and C



(a) Executable Spec. vs. Abstract Spec. (rho = 0.89, $p = 0.00$)

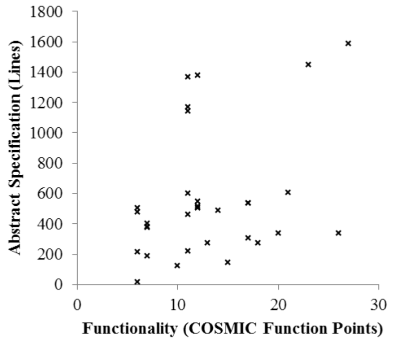


(b) Executable Spec. vs. C ($R = 0.95$, $p = 0.00$)

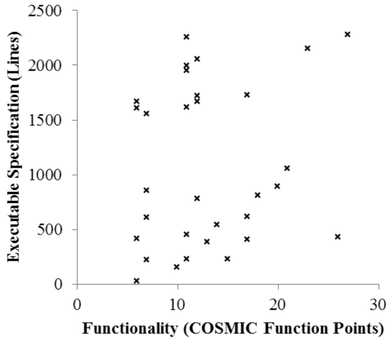


(c) Abstract Spec. vs. C (rho = 0.92, $p = 0.00$)

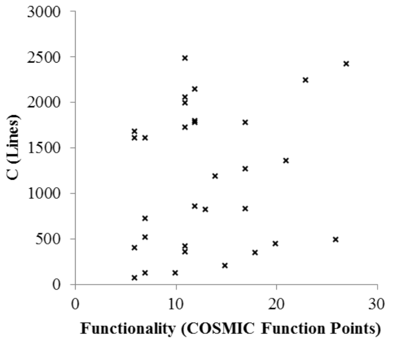
Fig. 1. Artefact Size Relationships. Each point is an seL4 API function.



(a) Functionality vs. Abstract Spec. (rho = 0.31, $p = 0.09$)



(b) Functionality vs. Executable Spec. ($R = 0.15$, $p = 0.42$)



(c) Functionality vs. C ($R = 0.19$, $p = 0.32$)

Fig. 2. COSMIC Function Point versus Artefact Size. Each point is an seL4 API function.

> 1500. The API functions in these top-most potential clusters are the same for the C and executable specifications, and are a super-set of those for the abstract specification. The top-most API functions in Figure 2a are exactly those which contain a call to a large complicated conditional recursive deletion sub-function within seL4. However, even after removing the contribution of this shared sub-function, and also manually removing one remaining outlier, we get a correlation of only rho = 0.37, with $p = 0.04$.

V. Discussion

Prior authors have found only weak-to-moderate relationships between CFP and source code line counts, and concluded that CFP should not be used as a predictor of source code line counts unless validated by local experiments [13]. For seL4, we found no significant relationship between CFP counts and C line counts. We also found no significant relationship between CFP counts and the line counts for formal specifications.

The most complicated internal function in seL4 is a recursive deletion function. Data management complexity and algorithmic complexity are not well addressed by CFPs [11], [16]. Our results support this, with low CFP counts for those APIs that call this recursive deletion function.

Although CFP counts were not well related to line counts, we did find very strong correlations between line counts for the formal specifications and code in the L4.verified project. Formal specifications would typically be defined later than FURs, because they are much more precise. Nonetheless they are defined relatively early in the project lifecycle, and so appear to be a promising target for size estimation.

A. Threats to Validity

Our results are from one project, and so we have no basis to claim they will generalise. Nonetheless our finding that CFP counts do not have a strong relationship to source code line counts is broadly consistent with prior studies.

A potential threat concerns the reliability of our CFP counts and line counts. One author is a measurement and estimation expert who has taught function point counting in industry, and has published research on function point methods. However, none of the authors are certified CFP measurers, and this study was the first time that the authors had performed CFP counting. A prior study [19] found that the reliability of CFP measurements increased with experience after conducting pilot studies. We partially addressed this threat by initial group tutorial studies of CFP, and by then conducting a pilot CFP counting exercise, as recommended, with independent counting followed by group review and discussion. This helped us to define and agree on the level of abstraction and objects of interest to be counted. We also addressed reliability by creating explicit CFP scripts of the CFP operations for each API function, and by having those scripts independently reviewed.

We have used CFP counts piecewise per API call, rather than summing them to create a system-level count, but this is consistent with a summary conclusion of the literature that function points should not be treated additively [11].

The line counting itself is straightforward and so its reliability is not a particular threat to validity, but the manual slicing of the specifications and code base into pieces reachable by each API function is a possible threat. After agreeing on an approach, the slicing of each project artefact was performed by separate researchers. These researchers then cross-checked their slices, confirming that their implementations were also consistent. As a further cross-check, we investigated outliers in the initial three-way scatter plots of relationships between line counts. This identified inconsistent counting of four API functions, because of errors in a few of the lists of functions contributing to the API slices. These were then corrected.

B. Related Work

There has been some prior work on size metrics and estimation for formal methods including size and complexity metrics for Event-B [20] and for algebraic specifications [21]. Prior authors have also proposed an automated calculation of function points (not CFP) from VDM-SL specifications [22] and B [23], but only compared manual and automated counts.

VI. Conclusions and Future Work

We studied size relationships between artefacts developed during the large-scale project that developed and formally verified the seL4 embedded systems microkernel. We found that CFP counts were poorly correlated with the size of implementation code and formal specifications. However, we did find strong and significant relationships between the sizes of the formal specifications and the implementation code. The Haskell executable specification, produced first in the project, were highly correlated with the size of the Isabelle/HOL abstract specification, produced second, and both of these were highly correlated with the size of the C source code, produced last.

Our overall goal is to increase the efficiency of formal verification projects through improved project management models and tools. The prediction of size of formal specifications from requirements is still an open problem. However, our next steps will be to investigate measures of the size and complexity of formal proofs in verification projects, and to examine their relationship to the size and complexity of formal specifications, source code, and system invariants. The creation of these formal proofs can be highly effort-intensive. Further future work is to investigate effort estimation models, perhaps based on architectural models of when proof work can be divided for independent concurrent work, and perhaps using process simulation techniques [9]. Replicating similar studies on other formal verification projects may provide a basis to generalise our findings to other project contexts.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," in *Proc. of 22nd SOSP*. ACM, 2009, pp. 207–220.

[2] J. Andronick, R. Jeffery, G. Klein, R. Kolanski, M. Staples, H. J. Zhang, and L. Zhu, "Large-scale formal verification in practice: A process perspective," in *Proc. of 34th ICSE*. ACM, 2012, pp. 1002–1011.

[3] J. P. Bowen and M. G. Hinchey, "Ten commandments of formal methods," *Computer*, vol. 28, pp. 56–63, 1995.

[4] S. Gerhart, D. Craigen, and T. Ralston, "Observations on industrial practice using formal methods," in *Proc. of 15th ICSE*. IEEE, 1993, pp. 24–33.

[5] J. P. Bowen and M. G. Hinchey, "Seven more myths of formal methods," *IEEE Software*, vol. 12, pp. 34–41, Jul. 1995.

[6] ISO/IEC 19761:2011, *Software engineering — COSMIC: a functional size measurement method*, 2011.

[7] J. Liedtke, "Toward real microkernels," *Communications of the ACM*, vol. 39, pp. 70–77, Sep. 1996.

[8] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[9] H. J. Zhang, G. Klein, M. Staples, J. Andronick, L. Zhu, and R. Kolanski, "Simulation modeling of a large scale formal verification process," in *Proc. of ICSSP*. IEEE, 2012, p. 10.

[10] *The COSMIC Functional Size Measurement Method Version 3.0.1: Measurement Manual*, The COSMIC Consortium, May 2009.

[11] C. Gencel and O. Demirors, "Functional size measurement revisited," *ACM Transactions on Software Engineering and Methodology*, vol. 17, pp. 15.1–15.36, 2008.

[12] J.-M. Desharnais, A. Abran, P. E. Dikici, M. C. Ilis, and I. N. Karaca, "Functional size of a real-time system," in *Proc. of IWSM 2009/Mensura 2009*. Springer, 2009, pp. 122–129.

[13] C. Gencel, R. Heldal, and K. Lind, "On the relationship between different size measures in the software lifecycle," in *Proc. of 16th APSEC*. IEEE Computer Society, 2009, pp. 19–26.

[14] Trustworthy Systems Team, *seL4 Reference Manual: API version 1.1*, NICTA, 2011.

[15] J. Rosenberg, "Some misconceptions about lines of code," in *Proc. of 4th METRICS*. IEEE, 1997, pp. 137–142.

[16] ISO/IEC 14143.5:2004, *Functional size measurement, Part 5: Determination of functional domains for use with functional size measurement*, 2004.

[17] L. Lavazza and C. Garavaglia, "Using function points to measure and estimate real-time and embedded software: Experiences and guidelines," in *Proc. of 3rd ESEM*, 2009, pp. 100–110.

[18] K. Lind and R. Heldal, "Categorization of real-time software components for code size estimation," in *Proc. of 4th ESEM*, 2010.

[19] O. O. Top, O. Demirors, and B. Ozcan, "Reliability of COSMIC functional size measurement results: A multiple case study on industry cases," in *Proc. of 35th SEAA*, 2009, pp. 327–334.

[20] M. Olszewska and K. Sere, "Specification metrics for Event-B," in *Proc. of 13th CONQUEST*, 2010, pp. 20–22.

[21] T. E. Hastings and A. S. M. Sajeev, "A vector-based approach to software size measurement and effort estimation," *IEEE Transactions on Software Engineering*, vol. 27, pp. 337–350, 2001.

[22] T. Miyawaki, J. Iijima, and S. Ho, "Measuring function points from VDM-SL specifications," in *Proc. of 5th ICSSSM*. IEEE, 2008.

[23] H. Diab, M. Frappier, and R. St-Denis, "A formal definition of function points for automated measurement of B specifications," in *Proc. of 4th ICFEM*. Springer, 2002.