Invited paper presented at the 3rd IEEE International Symposium on Requirements Engineering, Annapolis, MD, January, 1997. IEEE Computer Society Press, pp. 144–146.

Calculating with Requirements*

(Extended Abstract)

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

1 Automated Formal Methods

Requirements *elicitation* is concerned with discovering what is wanted; it necessarily depends on social processes such as discussion, introspection, review of documentation, and experimentation. Requirements *engineering*, on the other hand, is concerned with turning the products of elicitation into precise, unambiguous, and complete descriptions of what the system under consideration is to do. Although they shade into and complement each other, and may be part of a larger iterative process, requirements elicitation and engineering are different activities that need to be supported by different techniques and tools.

I maintain that *formal methods* provide techniques and tools that are appropriate—and effective—for the requirements engineering activity. They are effective because they allow certain questions about requirements to be reduced to calculations, and this is valuable because it allows *reviews* to be supplemented or replaced by *analyses*. I am using these terms in the sense in which they are employed in the guidelines for software on commercial aircraft [10, Section 6.3]: reviews are processes that depend on human judgment and consensus, while analyses are objective "mechanical" processes such as testing or calculation. Of course, certain questions do require human judgment, and some decisions require consensus, but many other issues are better addressed by analyses than by reviews: analyses are systematic, can be checked by others, and can even be automated. Especially when automated, analyses can be more reliable and thorough than reviews, and cheaper.

Formal methods are often advocated for the intellectual framework that they provide, and the methodological benefits that are believed to accompany their use. While I certainly agree that formal methods can help designers to conceptualize and formulate issues and requirements in a perspicuous and productive manner, I am skeptical that assurance based on human review is any more reliable for formal specifications than for informal descriptions. Instead, I claim that the distinctive benefit of specifically *formal* methods is that they support analysis through formal deduction—that is by theorem proving and related methods such as model checking. These are systematic processes that have the character of calculation and, like numerical calculations, they can be automated. The reasons for favoring mathematical modeling and calculation are the same in computer science as in other engineering disciplines: they allow the consequences of requirements and the properties of designs to be accurately predicted and evaluated prior to construction.

In most engineering disciplines, it is automation that releases the full potential of mathematical modeling: the highly efficient wings of a modern airplane could not be designed without the massive automation of computational fluid dynamics, finite element analysis, and several other mathematical modeling techniques. Automating the calculations underlying formal methods yields similar benefits for computer science: many questions that are currently examined by intensive but unreliable reviews, or by massive but necessarily incomplete testing, can be settled by automated calculation—thereby providing analysis that is more reliable and more comprehensive than at present, and liberating human reviewers for more creative and challenging tasks that truly require their judgment.

2 Experiments

Over the last few years, application of specialized but pragmatically effective theorem proving techniques, and of model checking and related methods, has made it possible to subject formal requirements specifications to several kinds of automated analysis. Some of these have been applied experimentally to require-

*This work was partially supported by NASA Langley Research Center under contract NAS1-20334, by the Air Force Office of Scientific research, Air Force Materiel Command, USAF, under contract F49620-95-C0044, and by the National Science Foundation under contract CCR-9509931.

ments documented in "Change Requests" (CRs) for the flight software of the Space Shuttle by a team involving the group at Lockheed Martin (formerly IBM) that develops this software, several NASA centers (Johnson, Langley, and JPL), and SRI. These experiments, running alongside what is generally considered an exemplary process for requirements review, provide useful anecdotal evidence for the effectiveness of automated formal analyses.

2.1 Very Strong Typechecking

One of the most basic "theorems" that can be proved about a specification is type-correctness. For programming languages, this theorem is "proved" by simple typechecking algorithms, but specification languages can use general theorem proving for this purpose with a corresponding increase in the strictness in the notion of type-correctness that can be enforced [9]. By simply describing its interfaces and functional requirements in such a very strongly-typed specification language, 86 issues (including 11 considered "major") were detected in several iterations of a CR for installing Global Positioning System (GPS) navigation on the Shuttle [4].[1]

2.2 Completeness and Consistency of Tabular Specifications

Rather stronger than typechecking are completeness and consistency checks for tabular specifications of the kind advocated by Parnas [14]. The idea is that the rows and columns of the table should partition the input space into distinct regions; if the circumstance in which each column (*resp.* row) applies is specified by a logical formula, then consistency requires that the pairwise conjunctions of the formulas should be false, and completeness requires that their overall disjunction should be true. Depending on the logic and theories used in the formulas, the deductive capabilities needed to discharge these proof obligations range from propositional tautology checking, though decision procedures for ground linear arithmetic, to full interactive theorem proving [11]. The capabilities at the lower end of this spectrum can be automated very effectively and can scale up to very large tables; those at the upper end are less automated and scale less well. Rapid progress is being made on developing balanced techniques that combine efficient and scalable automation with adequate expressiveness [1]. Applying some of these methods to a Shuttle CR for the "Heading Alignment Cylinder" (HAC), revealed that several rows in one table had overlapping conditions, and that several unstated assumptions needed to be articulated before the completeness and consistency of others could be established [3]. The Shuttle requirements analysts considered discovery of the missing assumptions to be particularly valuable. Similar successes with automated completeness and consistency checking have been reported for the RSML [6] and SCR [7] requirements methods.

2.3 State Exploration and Model Checking

Deeper analysis of tabular specifications is possible using model checking to examine whether certain desired properties are invariant or reachable. Atlee and Sreemani have demonstrated that such methods scale to quite large examples [13]. Model checking can also be used to perform analyses that are intermediate between testing individual cases (as can be performed by prototyping or animation), and proving general theorems. The idea is to "downscale" (aggressively simplify) the description of the system and its environment so that they become finite state. Finite state exploration or model checking methods can then be used to examine all the reachable states. Although the necessary simplification may be too aggressive for verification (i.e., properties of the simplified model may not be true of the full system) it is often adequate for refutation (i.e., bugs found in the model will also exist in the full system). Anecdotally, it seems that exploring *all* the states of a simplified system specification is much more effective at finding bugs than sampling those of the full system by prototyping or animation. These techniques are applied routinely (and very effectively) to hardware and protocols, but their application to software and to requirements is relatively new (see, for example [8]). Applying them to a Shuttle CR for three-engines-out abort sequencing revealed 19 errors, of which only 1 (albeit the most significant) had been discovered by human review [2].

2.4 Formal Challenges

Some of the most searching examinations of requirements specifications can be performed by "challenging" the specification with a putative theorem: "if this specification captures my intent, then the following ought to follow." Suppose, for example, that we had specified the operation of sorting a sequence; we might then challenge the specification by asking whether sorting an already sorted sequence leaves the

[1] At an earlier stage in this activity, formal methods had detected 30 issues, including 7 considered major, compared with 4, of which 1 was considered major, for the human review-based process [2].

sequence unchanged (i.e., we attempt to prove the theorem $\forall s : sort(sort(s)) = sort(s)$). Such a challenge would reveal a deficiency in many of the early specifications of sorting, which required the output to be ordered, but neglected to stipulate that it should also be a permutation of the input [5]. Applying this approach to the mature requirements specification for a Shuttle function known as "Jet Select" suggested the challenge "no jet that is designated as failed shall be selected for firing" [12]. The attempt to prove this was unsuccessful, revealing thereby a significant and previously undiscovered problem with the existing requirements specification.

3 Summary

Many issues in requirements engineering can be explored and analyzed using automated formal methods. At present, the tools supporting these analyses are not ideal: considerable knowledge and experience are required to select the most appropriate tool for a given task, to formulate the problem in a suitable manner, and to coax the tool into divulging a useful result. However, engineering challenges in the integration and scaling of formal methods tools are being rapidly overcome and these difficulties should soon be significantly reduced. Judicious use of automated formal calculations will allow many issues to be explored more completely and reliably than at present, and will allow the talents of human reviewers to be reserved for those problems in requirements analysis that truly require them.

References

Papers by SRI authors can generally be retrieved from `http://www.csl.sri.com/fm.html`.

[1] Clark Barrett, David Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201, Palo Alto, CA, November 1996. Springer-Verlag.

[2] Judith Crow and Ben L. Di Vito. Formalizing Space Shuttle software requirements. In *First Workshop on Formal Methods in Software Practice (FMSP '96)*, pages 40–48, San Diego, CA, January 1996. Association for Computing Machinery.

[3] Judith Crow and Ben L. Di Vito. Formalizing space shuttle software requirements: Four case studies. Submitted for publication, 1997.

[4] Ben L. Di Vito. Formalizing new navigation requirements for NASA's space shuttle. In *Formal Methods Europe FME '96*, volume 1051 of *Lecture Notes in Computer Science*, pages 160–178, Oxford, UK, March 1996. Springer-Verlag.

[5] S. L. Gerhart and L. Yelowitz. Observations of fallibility in modern programming methodologies. *IEEE Transactions on Software Engineering*, SE-2(3):195–207, September 1976.

[6] Mats P. E. Heimdahl. Experiences and lessons from the analysis of TCAS II. In Steven J. Zeil, editor, *International Symposium on Software Testing and Analysis (ISSTA)*, pages 79–83, San Diego, CA, January 1996. Association for Computing Machinery.

[7] Constance Heitmeyer, Alan Bull, Carolyn Gasarch, and Bruce Labaw. SCR*: A toolset for specifying and analyzing requirements. In *COMPASS '95 (Proceedings of the Tenth Annual Conference on Computer Assurance)*, pages 109–122, Gaithersburg, MD, June 1995. IEEE Washington Section.

[8] Daniel Jackson and Craig A. Damon. Elements of style: Analyzing a software design feature with a counterexample detector. *IEEE Transactions on Software Engineering*, 22(7):484–495, July 1996.

[9] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[10] *DO-178B: Software Considerations in Airborne Systems and Equipment Certification.* Requirements and Technical Concepts for Aviation, Washington, DC, December 1992. This document is known as EUROCAE ED-12B in Europe.

[11] John Rushby. Mechanizing formal methods: Opportunities and challenges. In Jonathan P. Bowen and Michael G. Hinchey, editors, *ZUM '95: The Z Formal Specification Notation; 9th International Conference of Z Users*, volume 967 of *Lecture Notes in Computer Science*, pages 105–113, Limerick, Ireland, September 1995. Springer-Verlag.

[12] John Rushby. Automated deduction and formal methods. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 169–183, New Brunswick, NJ, Jul/Aug 1996. Springer-Verlag.

[13] Tirumale Sreemani and Joanne M. Atlee. Feasibility of model checking software requirements. In *COMPASS '96 (Proceedings of the Eleventh Annual Conference on Computer Assurance)*, pages 77–88, Gaithersburg, MD, June 1996. IEEE Washington Section.

[14] A. John van Schouwen, David Lorge Parnas, and Jan Madey. Documentation of requirements for computer systems. In *IEEE International Symposium on Requirements Engineering*, pages 198–207, San Diego, CA, January 1993.