Experiments Suggest High-Level Formal Models and Automated Code Synthesis Significantly Increase Dependability

Cordell Green and Stephen Westfold January 2001

Kestrel Institute

Abstract: We discuss two experiments that suggest error rates might be reduced by factors of 2-20 by using formal high-level models and correct-by-construction automated software design. One high security application provides further evidence that these methods can produce highly dependable software.

Dependencies viewed as a major source of software error: The complexity of large software applications and the processes that create them is outgrowing our current tools, technology and processes. Evolving requirements necessitate rapid growth in the number of functions that must be created in software. As this increasing software functionality grows, the software to carry out these functions also grows at a daunting rate.

More challenging to large projects is the frequent issue that the practical software difficulty grows yet faster than the software size itself. Restated, the cost and schedule grow faster than application size [Boehm]. In particular, their growth is proportional to size raised to an exponential power (typically near 1.2). Or expressed in terms of error potential, the chance of mission-critical errors can grow faster than size, and one sees a difficult tradeoff between schedule, cost, or chance of critical errors.

A key source of the problem is the number of dependencies in the software; each time a module is (re)integrated into a system it can interact with other modules. Each interaction or potential interdependency with other modules results in a potential for error if each interdependency is not handled adequately.

It becomes very difficult to keep all the software consistent and determine the impact of all changes when each new increment or change in an application can potentially have an impact in many places in the software. Experiments show that current best practices can reduce the superlinear error difficulty by reducing the exponent from ~1.2 to ~1.1. Where else can we turn for additional help?

Experiment One: Comparing Dependencies in Specs to Dependencies in Code: We performed a simple experiment that suggests the use of high-level formal models and automated code synthesis might reduce errors by a factor of 2-5.

We focused on a common source of software errors, namely dependency errors. Typically, a change or addition is made to the software, but its effects on other parts of the software are not properly handled, introducing an error. An example would be where one module assumes distance is measured in miles, but another module assumes distance is measured in kilometers. These errors can occur at the unit level, but are especially

challenging at the integration level. We make the oversimplified but not unreasonable assumption that such dependencies are a major source of errors.

We were able to carry out this experiment by virtue of having an automated code synthesis system capable of provably correct construction of code from formal models (formal specs). For each of three software applications we simply counted the number of dependencies in the high-level formal model, as expressed in Kestrel's Specware specification and refinement system, and then also counted the number of dependencies in the code synthesized from the formal model(specification). In this short note we will not delve into the method of measuring dependencies. The three applications were a simple sorting program, a highly optimized chess problem, and a larger transportation scheduling application.

The number of dependencies increased by a factor of two to five in the translation from specification to efficient code. Software engineers are familiar with the various ways this happens, including introduction of temporary variables, information spreading, low-level data structures, complex optimizations, etc.

One of the authors performed a "hand recount" of the dependencies to verify that they accurately reflected dependencies that a software engineer would need to take into account at each of the levels (high-level model versus low-level code)

The assumption that the major source of errors is dependency errors implies a potential error reduction by roughly the same factor of two to five. This is greatly oversimplified, not taking into account such factors as other sources of errors or the accuracy of developing high-level models. Another issue is that the formal tools are today just emerging research prototypes, not commercial-level tools. Still, this experiment suggests a more comprehensive experiment is called for, taking other kinds of errors into account, correlating errors with dependencies, and exploring whether simple high-level models might reduce other errors such as logic errors.

One might question why we should trust the code synthesis machinery. One answer is simple, namely that the formal methodology permits machine proofs of correctness. An economic argument is that capital investment in the automated synthesis/translation tools is recouped in the many re-uses of the translator. That is, put your effort into the translator and reuse that effort many times, analogously to the investment in a trusted compiler.

Experiment Two: Comparing Conventional Practice to Formal High-Level Modeling: The National Security Agency performed an experiment[Widmaier] comparing two methodologies: high-level formal modeling including automated code synthesis versus current best practices. NSA contracted with two large corporations to create a software application, a secure card entry system. One team used Kestrel's Specware high-level formal modeling and automated synthesis tool to formally specify and refine the model into code. The other team used UML and current practices,

generating code by hand, and was rated at SEI's software management capability maturity model level 4(CMM4, the second-highest level).

Both teams were given the same requirements document and were allowed the same funding and time to design and code the system. Neither team knew of the other, nor that they were part of an experiment; Kestrel was also not aware of the experiment. NSA employed the University of Maryland to test the two resulting software systems on a large test suite and then additionally employed an auditing company to verify the university's results.

On critical errors, the team using conventional methods scored 56 percent correct and the team using formal modeling and automated synthesis scored significantly higher, 75 percent correct. Especially interesting was the observation that if a single change were made by the formal modeling team, namely their interpretation of a timing requirement, their score would be improved to 98 percent correct(i.e., 46% critical errors vs 2%). The timing specification was in a single location in the formal model, so that it could be readily changed if agreement on interpreting the requirements was reached. In contrast, there appeared to be no simple fix to the errors in the conventionally-produced software, since the errors were relatively uniformly distributed throughout the code modules. This second experiment thus further suggests promise and the need for a larger experiment.

Further Validation via Commercial Development of Software: Additional validation of the proposed approach is suggested by Motorola's use of the high-level formal modeling and automated synthesis approach in developing a secure operating system separation kernel for their AIM security system, now deployed on the F-22 aircraft. This OS separation kernel enables multiple users with different security access levels to share a computer, not needing physically separate machines. The technology allowed the precise mathematical characterization of the separation and also enforced the accurate synthesis of an implementation faithful to the specification. We understand that this was the first version of this application to receive certification from NSA.

References

Boehm, Barry W., *Software Cost Estimation with Cocomo II.* Prentice Hall, Upper Saddle River, N.J., 2000.

Widmaier, J.C.; Smidts, C.; Xin Huang, Producing more reliable software: mature software engineering process vs. state-of-the-art technology? Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium, p.88-93. New York: ACM Press, 2000.