Redundancy on the software design process is essential for designing correct software

Redundancy in the Software Design Process is Essential for Designing Correct Software

by Mark G.J. van den Brand and Jan Friso Groote

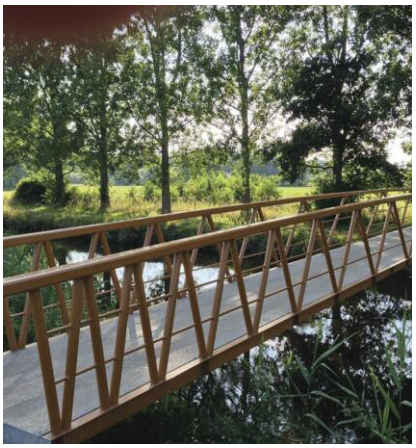
Researchers at Eindhoven University of Technology in the Netherlands plead the case for more redundancy in software development as a way of improving the quality of outcomes and reducing overall costs.

If an engineer was asked how reliable a critical artefact must be, a likely answer might be that a system can only fail once in 10^{10} times. For example, the probability of a ship's hull collapsing during its lifetime or an active high water barrier failing to function when it is requested to do so, are typically in the order of 10^{-10}. These numbers are so low that no engineer will ever experience the failure of his own artefact.

Considering such a question led us to reflect on how it may be possible to obtain similar numbers when designing software. In our previous work, we addressed this question by constructing a simple failure model and, we found a simple answer [1]. The only way to obtain such figures was by employing redundancy. Such an approach is very common in the more classical engineering disciplines. For example, when considering support columns with a failure probability of 10^{-5}, an engineer can simply use two columns (where only one is truly necessary), thus allowing the overall failure probability to be increased to 10^{-10}.

To examine how this redundancy approach applies in the software development field, we must realise that software is very different from physical artefacts. In physical artefacts, components fail due to wear and tear while software fails due to built-in errors, for example, a small typing error, an incorrect algorithm or the wrong use of an interface. When such an error is activated, the software fails. As software has many varied states, it can take a long time for some errors to become active, although as shown in [2] many programming faults lead to a multitude of erroneous states. Therefore, these latter faults are far easier to catch.

The probability of a hardware failure is almost negligible and thus, can be ignored. Software errors are always directly caused by either the programmers or program designers that left those errors in the code. As humans they have a large probability of doing something wrong [3]. At best, the failure probability of humans is only 10^{-3} but even this figure can only be applied in situations where the tasks are very simple and the programmer highly trained. For more complex tasks, failure probabilities of 10^{-2} or 10^{-1} are more realistic. In situations where a human must complete a non-trivial task under stress, they are almost certain to fail.



This footbridge (Nederwetten, The Netherlands) is made of more steel than strictly necessary to assure its quality. Software engineers must also consciously employ redundancy to ensure quality.

It should be obvious that the difference between the failure probability of a programmer and the desired failure probability of a critical piece of software is around eight orders of magnitude. Obvious measures, such as comprehensive training for programmers or the use of the most modern programming languages are excellent solutions but alone, these measures are unable to bridge this gap. Training can never accomplish an improvement of more than a factor 10^{-2} and for a complex task such as programming, even this is unlikely. Using modern programming languages, even domain specific languages, in combination with libraries can lead to substantial reductions in the amount of required code and thus, reduce the overall numbers of errors. However, here too, the possible reductions that can be achieved (at most a factor of 100) are insufficient.

Thus, the only way to achieve the desired failure probability of 10^{-10} is to consciously employ redundancy in the software design process. Typically, when constructing software, it must be described in several ways. These differing approaches should then be meticulously compared and challenged, with the goal of removing as many of the flaws that will be inherent in each description.

Several forms of redundancy are already present in actual programming, such as type checking and testing. However, these forms of redundancy came about as good practices, not conscious ways to introduce redundancy with a view to attaining a certain level of software quality.

Active redundancy can be brought into the software design process through the introduction of high level models of the software, for instance, in the form of domain specific languages, property languages such as modal logics to independently state properties, independently (and perhaps multiple) constructed implementations, and a priori described test cases. The comparison of these different views can be done by model checking (software or models against properties), model based testing (model against implementation), and systematic testing (tests against model or software). Code inspection and acceptance tests are also fruitful, but lack the rigour of comparison that the more mathematical methods have.

By acknowledging that redundancy in design is the only way to obtain reliable software, one can then question certain trends. For instance, there is an ongoing trend to eliminate the annoyance associated with static type checking. A language like Python is a typical

example, as is the introduction of the auto keyword in C++ which allows a programmer to skip writing down an explicit type. The desire for code generation out of a model, or research in generating a model and software out of requirements put on the software introduce a single point of failure in the design process. These approaches do not pay tribute to the need for redundancy and discourage the detection of flaws that are inevitably made in the design of the software. This is a serious problem, as we all know that such flaws can wreak havoc when they happen to be activated.

Links:
http://www.win.tue.nl/~mvdbrand/
http://www.win.tue.nl/~jfg/

References:
[1] M.G.J. van den Brand and J.F. Groote: " Software Engineering: Redundancy is Key", J.J. Vinju editor, preprint Science of Computer programming, Special issue in Honour of Paul Klint, pp. 75-82, 2013.
[2] J.F. Groote, R. van der Hofstad and M. Raffelsieper: "On the Random Structure of Behavioural Transition Systems", Computing Science Report CS-R1401, Department of Mathematics and Computer Science, Eindhoven University, 2014. http://www.win.tue.nl/~jfg/articles/CSR-14-01.pdf
[3] D.J. Smith: "Reliability, maintainability and risk. Practical Methods for Engineers", Elsevier, 2011.

Please contact:
Mark G.J. van den Brand
Eindhoven University of Technology, The Netherlands
Tel: +31402472744
E-mail M.G.J.v.d.Brand@tue.nl

Jan Friso
Eindhoven University of Technology, The Netherlands
E-mail J.F.Groote@tue.nl

Estimating the Costs of Poor Quality Software: the ICEBERG Project

by Luis Fernández, Pasqualina Potena and Daniele Rosso

Project ICEBERG investigated a novel approach to improving understanding of the real cost impacts of poor quality software and supporting the suite of management decisions required to take corrective action across the entire software development cycle.

The ICEBERG project was developed to consider the issue of Transfer of Knowledge (ToK) in the Software Quality Assurance (QA) domain and had two main objectives: (1) investigating, defining and implementing model-based processes oriented to identifying the most effective and efficient QA strategy for software development in general, and more specifically, software developed for telecommunications and finance organisations; and, as stated for this type of Marie Curie projects, (2) bolstering the research platform in this area for future work through the secondment of researchers and the specific training of early stage and recruited researchers.

Project Motivation

Commonly, software projects need to be performed and delivered against project schedules that specify timings, costs and quality constraints (amongst other things). One of the most cost- and time-intensive components of the overall development cycle is the QA process. A major issue associated with this process is that the individual analysis of single factors in isolation is frequently inaccurate, as pairs of factors may visibly (and sometimes adversely) affect each other. Therefore, frameworks that support decisions made in relation to meeting scheduling and quality requirements, while keeping project costs within budget, would be very helpful for project managers.

Research Themes and Challenges

The ICEBERG project started in February 2013 and will end in December 2017. It is funded through the European Marie Curie program (IAPP category). The project's main scope is to provide researchers with new research skills and broaden the horizons of models-based processes with a view to identifying the most effective and efficient QA strategy in software development.

A number of institutions collaborated on this project: two research centres (CINI (Consorzio Interuniversitario Nazionale per l'Informatica) - University of Naples and University of Alcalá (UAH)) and two SMEs (Assioma.net and DEISER). Specifically, the two universities provided skills in the areas of quality estimation and forecasting models of software products/processes and related costs. The SMEs contributed highly qualified real-life experience on the testing of software projects/processes. The project involves up to 19 researchers who all have the opportunity to make cross-entity swaps with the other partner institutions. The researchers then have the opportunity to share their capacities, acquire new skills and develop new competences on decision support systems in the quality assurance domain. Once they return, this knowledge flow continues, this time back to their home institutions, enhancing European economic and scientific competitiveness. Up to three researchers have been specifically contracted for periods of 18 or 24 months in order to contribute to the project and to be trained as specialists in the field..

The key focus of the project will be the enhanced support that a joint analysis of schedules/times, costs and quality can