

A Safe Change Oriented Process for Safety-Critical Systems

Nigel Tracey Alan Stephenson John Clark John McDermid

Department of Computer Science.

University of York,

Heslington, York.

YO1 5DD, England.

+44 1904 432749

{njt, al, jac, jam}@cs.york.ac.uk

Abstract

The development of aviation engine control software is a prime example of a process in which the overriding concern is safety. In building such a complex system, change is inevitable. This paper discusses the aims of CONVERSE, an EPSRC-funded project to investigate change management in safety-critical software development. The aim is to reduce the costs which occur due to change, while allowing rapid re-collection of the safety evidence required for such systems.

1 Introduction

The Engineering and Physical Sciences Research Council (EPSRC) has funded a number of research projects under the title ‘Software Engineering for Business Process Change’ (SEBPC). This paper describes the research of the CONVERSE project which is part of the SEBPC programme. CONVERSE’s focus is the development process for Full Authority Digital Engine Controllers (FADEC) for aviation engines and is being carried out in collaboration with Rolls-Royce.

The FADEC is a complex hydro-mechanical system, but the majority of its functions are implemented in software running in an electronic engine controller (EEC). In practice there are different FADECs and EECs for different engines. More precisely, different series or types of engines are produced, e.g. the Trent and the BR700. Within these series, there are engine marks typically representing different power ranges, e.g. Trent 700 and Trent 800. In addition each mark may have a number of variants usually according to the airframe on which the engine is to be installed, e.g. Airbus A340 and Boeing 777.

The objectives of CONVERSE are to demonstrate a new process in which a *standard EEC* is offered, which is modified for particular variants, families and customers. Hence, the focus of the new process becomes change and the management of that change. CONVERSE is addressing this problem in two ways. Firstly, domain analysis and domain modelling are being used to identify commonality and variability. This is combined with the use of design and implementation architectures that aim to localise the effects of change. Secondly, techniques for automated re-verification and re-validation are being developed. These techniques will focus on providing the evidence required to regain confidence after a change.

The organisation of this paper is as follows. Section 2 summarises current FADEC development and outlines the techniques required to aid change. Section 3 discusses the process and life-cycle of change in the context of a safety-critical system. Some preliminary ideas on designing to accommodate change are presented in section 4. Section 5 discusses techniques for re-verification in response to change. The paper concludes with a presentation of some of the research issues forming the basis for the CONVERSE project.

2 FADEC Development

The production of a new FADEC within Rolls-Royce is effectively a bespoke development despite the commonality of key functions. This is mainly because of the constraints placed on the development process associated with meeting the relevant standards, such as DO-178B[2]. These standards are imposed because of the overriding concern for safety of the final engine controller system. While an engine is in use, fuel, air and oil all flow through it at high speeds, temperatures

and pressures. These must all be carefully controlled to ensure that no part of the engine is operating outside of its design limits. An essential of the development process is the collection of safety evidence. The techniques currently used to obtain this evidence are generally applied to the whole engine.

Change can occur throughout the development process, and each change has the potential to compromise the safety evidence which has already been collected. Due to the *whole engine* approach used to collect evidence, the easiest way of dealing with change, currently, is to re-analyse the whole engine control system. Generating a new, complete and consistent set of evidence. This is obviously extremely inefficient and costly.

In order to reduce the cost of re-analysis, we propose to introduce techniques which can isolate the area affected by a change, and which then only perform re-analysis and re-verification on this localised part of the software. Rather than introducing these ideas incrementally, however, we aim to facilitate the introduction of a radical process change.

3 Change Process

3.1 Understanding Change

Change occurs in the development process for a variety of reasons; here, we list some types of change which we expect to be able to address:

Misunderstanding There are many techniques which attempt to reduce the potential for misunderstanding. These techniques typically try to form a common consensus with agreed terminology, scope and semantics. The potential for misunderstanding in this situation is therefore higher when the opportunity for consensus is lower — as is the case between large organisations.

Implicit assumptions Whenever a system is described, there are always assumptions made about the context in which that system will operate. While it is impractical to document every aspect of that context, it is possible to describe some of the more important assumptions, such as those affecting safety. When an implicit assumption turns out to be wrong, it is potentially very difficult to identify those parts of the system which were dependent on that assumption.

Emergent re-design System design, especially that of software systems, is made with reference to some specification. Not all of the properties of the system can be predicted directly from the design — the so-called emergent properties. If a sys-

tem's emergent properties do not meet the specification, then the system must be re-designed, or the specification must be re-negotiated.

3.2 The Introduction of Change

In order to effectively identify the impact of a change, the software engineer must be able to track that change from its entry into the development process. For example, a change in certification requirements affects the gathering of safety evidence, which in turn could place additional constraints on the implementation methods used, or even on the software architecture. The software development model must be able to represent this kind of dependency.

Another important aspect of the introduction of change is the dependencies between the changes themselves. Some changes might only make sense when considered as a group of changes, rather than individually. Indeed, it may be possible to exploit the properties of a recurring group of changes as a pattern.

3.3 Anticipating Change

Several successful change management techniques have made use of family-based analysis, identifying features common to all of the systems in a family, and specifying variabilities to accommodate the variation within the family[1]. This technique can be used to anticipate changes within the family, by extending the variability to cover likely scenarios in the near future. Engine control software is a good example of a such a family, and so a technique which makes use of this commonality and variability analysis should work well within this domain.

Once a description of previous and anticipated changes is available, the designer can use this information to construct designs which are resilient to the types of change described. Parts of the system which vary greatly could be decoupled from the common parts, in such a way as to prevent the propagation of expected changes, for example, across the decoupling connection.

3.4 Assessing Change Impact

The key aspect of the process must be the identification of the elements of the system which are affected by the introduction of a particular change. Given the right information about the interactions between those elements, it should be possible to follow the dependencies from the original location of change to all of the affected parts. It should also be possible to describe how they are affected, which assists in responding to the change. Tools which help to perform this analysis, or even which automate it, would also reduce the cost of this stage.

3.5 Responding to Change

Whenever there is a change introduced into a system, there is the possibility that the system must respond to that change. The techniques outlined above would certainly help to identify the areas in which this response would take effect. The response can vary from a situation where the system needs no modification to cope with the change, through expected changes which have been planned for, right up to completely unexpected circumstances which require a large planning and re-engineering effort.

Ultimately, the response will be some combination of the following:

Insertion A new element is added to the system to cope with the change.

Removal An element, which will not be needed because of the change, is removed.

Replacement An element is replaced with one which performs an equivalent function in the new circumstances.

Re-parameterisation Some parameters, such as a type or a range, are changed to match the new situation.

Reconfiguration If elements are changed, it is likely that a part of the application must be reconfigured to allow their operation.

The response itself could generate further changes, to which the system must again respond. This can be addressed by repeatedly performing the impact analysis until no more response is required, but this is, in general, still costly and inefficient. This situation would benefit from designs which can respond to certain types of change without themselves propagating further change.

Once the system has responded, it must be re-analysed. In order to make this process as rapid as possible, only those elements which have been affected by the change should be analysed. For this, the dependencies used by the impact analysis techniques must be those which can show the effect of a change on the sources of safety evidence. The impact of the change must cover all of the systems which would need to be re-analysed in order to create valid and consistent safety evidence.

4 Designing to Accommodate Change

It has been noted that impact analysis would benefit from designs which can respond to a change without themselves generating and propagating changes.

It would be difficult to account for every possible type of change, but an analysis could be made of the likely types of change, and their impact on the collection of safety evidence. For example, if it is expected that floating-point numbers would be used instead of, or even alongside, fixed-point representations, then the elements more resistant to this change would be those which do not make assumptions about the numerical representation which were true of fixed-point numbers but not of floating-point numbers.

This clearly relates to the software engineering issues of architecture and patterns. An analysis of anticipated change could indicate patterns of elements which were suited to the management of those types of change and the analyses which must be performed in the light of such changes.

5 Re-verification

A key aspect in the management of change is re-verification. The requirement to produce *flight-standard* software conforming to DO178B requires the collection of safety evidence for the software which runs on the EEC. Testing is a significant source of this evidence, indeed DO178B imposes a number of requirements on the testing activities (e.g. full modified condition/decision coverage). Typically, at least 50% of the development costs are consumed by testing activities. Any process which is to be successful must therefore be able to tightly bound the re-verification effort required in the light of a change. In response to a change the aim is to quickly and cheaply return to the same level of confidence in the software with regards to functionality and safety.

The major focus of the work to date has been the automatic generation of test-data. A framework based on the application of heuristic global optimisation techniques has been developed. This framework has been targeted at a number of testing problems relevant to re-verification. A brief outline of these applications is given below, full details can be found in [10, 6, 7, 8].

5.1 Specification Conformance

A software unit's specification can be represented as a pre-condition and post-condition. The most interesting test-data is that which illustrates the specification has been broken. What is required is test-data which meets the pre-condition but breaks the post-condition after execution of the software under test (SUT). A cost function has been developed which allows the optimisation framework to search for test-data which meets this requirement. Such test-data is clearly important for re-verification. After any change to the software or specification it is important that

any specification conformance violations of the software are found.

Optimisation techniques themselves cannot guarantee their results. Indeed as with any testing technique only the presence of faults can be shown not their absence. This is important as a failure of the optimisation based test-data generation does not imply the SUT is correct with respects to its specification. However, the test-data generation performs an aggressive targeted search and therefore failure to locate a specification violation does allow increased confidence in the validity of the SUT. This test-data generation process is completely automatic. This technique could be integrated into the change process, such that in response to a change it is used to quickly (and cheaply) gain confidence that the change has not broken the specification. Traditional techniques, such as proof or extensive testing, could then be used. The specification being used as the basis for the test-data generation need not be a full functional specification. Indeed, safety conditions or system invariants could be used instead.

5.2 Structural Coverage

Structural coverage requirement placed on testing by all of the safety-critical development standards, such as DO178B [2], 00-55 [4] and IEC 61508 [3]. Typically 25% of the total development costs are spent on devising tests to meet this requirement. However, the metrics collected on the testing process shows that this form of testing does not find any serious defects that are not identified by other forms of testing. However, as structural testing is a requirement it has to be performed. The test-data generation framework has been targeted at this form of testing. The developed cost function allows the optimisation technique to search for and locate data which executes a specified statement or path. Ideally close to full coverage should be obtained using requirements based testing. The test-data generation framework can then be used to fill in gaps in coverage to meet the certifications standards. This process could reduce the costs required to develop structural test-data and allow new test-data to be generated quickly in response to changes. The reduction in costs may allow the testing effort to be better invested in other forms of testing.

5.3 Assertion/Exception Conditions

Assertion checks are constraints which the developer has specified must hold at a particular point in the code. Exception conditions are assertions which must hold to ensure the run-time rules of the language are not violated (for example, numeric overflow or out of range errors). Targeting testing at causing

specific exceptions or assertion conditions may be important. The run-time checks associated with checking for exception conditions make the process of gaining full test coverage very much more difficult. Typically in safety-critical systems the run-time checks are disabled in the production system based on the assumptions that exceptions do not occur. The test-data generation framework can be used to target test-data which causes exceptions or violates assertions. This technique can be integrated with tools such as the Spark Examiner [5] to reduce the costs involved in proving exceptions cannot occur and that assertions will not be violated. Full details on the technique can be found in [9].

5.4 Reuse Testing

When re-using components it is vital that the environment within which they are re-used meets the assumptions that were made when the component was developed, i.e. for all input-data in the component's domain the pre-condition holds true. The goal of reuse testing can be stated as finding test-data which causes the precondition of a re-usable component to be broken. The optimisation search is guided to test-data which executes the call of the component and also breaks the pre-condition of the component. In this way unsafe reuse of a component can be illustrated.

6 Research Proposal

The overall aim of the project is to enable the creation of a software development process geared towards the construction of safety-critical software under change. This process would build on some of the recent work on requirements notations within the company, which is based around a lexicon- and table-based requirements notation, and identifies the relationships between design and requirements elements at successive layers of abstraction.

The lexicon would be extended into a complete domain model, to facilitate the identification of those aspects of commonality and variability which drive the changes in the software development. The ultimate aim is for the company to maintain a domain model which covers all types of safety-critical engine control systems.

The ability to enable systematic reuse is seen as a good way of reducing recurring costs at the same time as increasing confidence in the software product. It is expected that features of a change-based development process will work together with a systematic reuse system, especially in the management of test cases. While enabling systematic reuse, it is vital that such reuse does not compromise the safety of the system. Verification techniques related to reuse have been

prototyped. These will be extended and targeted at the safety aspects of reuse. The aim is to examine the feasibility of these optimisation-based test automation tools. Using real engine controller code, the scalability of these techniques to industrial problems will be assessed.

An investigation into the expected types of change is needed in order to create designs which are resistant to those types of change. There is a trade-off here; designs resistant to some changes could be vulnerable to other types of change, and so the success of this analysis is crucial to change management.

The design of a set of software elements which are resistant to some types of change, and which can be analysed for the effect of a change on their safety properties, clearly requires a well-defined domain-specific software architecture. It is expected that this architecture would be in some way conformant with the structure of the commonality identified in the domain model, and it would also take into account the types of change expected.

Testing effort consumes a large proportion of both the recurring and non-recurring costs involved in developing a safe EEC. The focus is to automate as much of the testing effort as possible, particularly in the areas of test-case construction and test-data generation. To some extent these issues have been addressed in the authors' previous work. What is required now is the application of the techniques to a real system. These case-studies are currently being carried out. Test-case management is also an important aspect. In response to change, efficient identification is required of all tests to be rerun and invalidated tests. The optimisation-based framework developed to date will be extended to address these higher level testing issues.

The complete software engineering process would be validated on a case-study, implementing a wide cross-section of the functionality present in an engine control system. If successful, this would lead the way towards a radical process change within the company.

7 Acknowledgements

This work was funded by grant GR/L4872 from the Engineering and Physical Sciences Research Council (EPSRC) in the UK as part of the SEBPC project. It builds on work started under grant GR/K63702 also from the EPSRC. The authors would also like to thank Rolls-Royce for their support.

References

- [1] J. Coplien, D. Hoffman, and D. Weiss. Commonality and Variability in Software Engineering. *IEEE Software*, 15(6):37–45, November 1998.
- [2] Radio Technical Commission for Aeronautics. RTCS/DO-178B: software considerations in airborne systems and equipment, December 1992.
- [3] IEC. 61508 – functional safety of electrical / electronic / programmable electronic safety-related systems. International Electrotechnical Commission, Draft Standard, December 1997.
- [4] MoD. 00-55 requirements of safety related software in defence equipment. Ministry of Defence, August 1997.
- [5] Praxis Critical Systems. *Spark-Ada Documentation 2.0*, 1995.
- [6] Nigel Tracey, John Clark, and Keith Mander. Automated program flaw finding using simulated annealing. In *International Symposium on Software Testing and Analysis*, pages 73–81. ACM/SIGSOFT, 1998.
- [7] Nigel Tracey, John Clark, and Keith Mander. The way forward for unifying dynamic test case generation: The optimisation-based approach. In *International Workshop on Dependable Computing and Its Applications*, pages 169–180. IFIP, 1998.
- [8] Nigel Tracey, John Clark, Keith Mander, and John McDermid. An automated framework for structural test-data generation. In *Proceedings of the International Conference on Automated Software Engineering*. IEEE, October 1998.
- [9] Nigel Tracey, John Clark, Keith Mander, and John McDermid. Automated test-data generation for exception conditions. Technical report, Department of Computer Science, University of York, 1999.
- [10] Nigel J. Tracey. Test-case data generation using optimisation techniques – first year DPhil report. Department of Computer Science, University of York, 1997.