2014 IEEE 15th International Symposium on High-Assurance Systems Engineering

On the Nature and Content of Safety Contracts

Patrick Graydon[1] and Iain Bate[1,2]
[1]Mälardalen University, Västerås, Sweden
[2]University of York, York, UK
patrick.graydon@mdh.se, iain.bate@cs.york.ac.uk

***Abstract*—Component-based software engineering researchers have explored component reuse, typically at the source-code level. Contracts explicitly describe component behaviour, reducing development risk by exposing potential incompatibilities early. But to benefit fully from reuse, developers of safety-critical systems must also reuse safety evidence. Full reuse would require both extending the existing notion of component contracts to cover safety properties and using these contracts in both component selection and system certification. In this paper, we explore some of the ways in which this is not as simple as it first appears.**

***Keywords*—CBSE, safety, contracts, modular safety case**

I. Introduction

Software engineering researchers have extensively explored the reuse of components' implementations, usually at source-code level. Researchers in the Synopsis project propose leveraging Component-Based Software Engineering (CBSE) to lower the cost of developing safety-critical systems [1]. In CBSE, developers describe software components using contracts [2]–[4]. Each contract describes a *guarantee* that can be made about a component's behaviour provided that an *assumption* is satisfied. But to benefit fully from reuse, developers of safety-critical systems must also reuse as much safety evidence as practicable. This requires both extending existing the notion of component *contracts* to cover safety properties and using these contracts in both component selection and system certification. This is not as simple as it first appears.

II. The Definition and Role of Safety Contracts

Researchers in the SafeCer project [1] propose using safety contracts to capture the safety-relevant behaviour of software components used in safety-critical systems [1], [5]. A *software component* is an identifiable software unit that communicates through explicitly specified interfaces. *Component types* are the reusable, non-system-specific form of components and might be packaged and distributed as either source or object code. *Component instances* comprise distinct portions of the system's design, the software source code, and the object code that will be deployed as part of a complete system. Researchers have given the following examples of what contracts for component types and component instances might specify [6]:

- [Input] X must be an integer in the range [0, 100]
- If X is in [0, 10], then [output] Z is in [-∞, 50]
- Output X is always less than the sum of inputs Y and Z
- Calls to a component that reads and writes files must follow the sequence (Open; (Read | Write)* ; Close)*
- The maximum dynamic memory usage is 100 bytes
- The WCET of the provided service A is 150 milliseconds
- Value errors on input port B do not affect output C

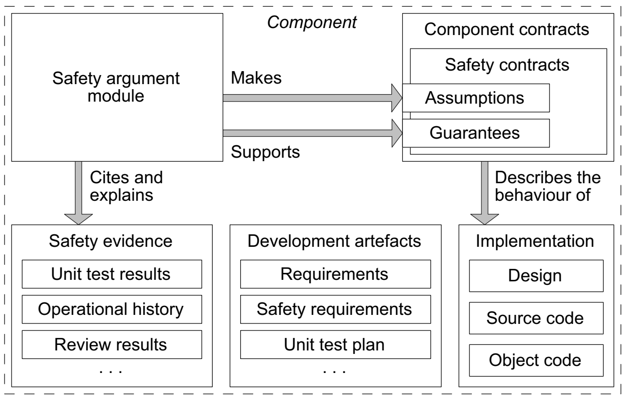

Fig. 1. Relationship between components, safety evidence, and contracts

Many of these properties could be the subject of both safety contracts and non-safety contracts. The distinction is whether or not the guaranteed property is needed to manage a hazard.

In the SafeCer vision [1], [4], safety contracts would help to modularise safety evidence, facilitating qualification of components. Fig. 1 illustrates the relationship between safety contracts and components. Fig. 2 illustrates the lifecycle for safety-critical CBSE that researchers propose [5]. Component types might be created either for a specific system or out of context. Reusable components would ideally be qualified: that is, assessors would check the evidence cited in support of component types' contracts so that these properties can be assumed when certifying systems using those components.

The lifecycle illustrated in Fig. 2 makes a safety contract play four roles: (1) a means of encapsulating portions of the safety case; (2) a target for component type design; (3) a placeholder to facilitate system design; and (4) an indicator of expected performance. Some of these roles are more crucial to safety than others. As we will, creating contracts that perform all of these roles is not straightforward.

III. The Devil in the Details

In this section, we present example safety contracts, discuss challenges to making contracts useful for all of the roles they will play in the SafeCer process, and suggest improvements.

A. Nominal Functional Behaviour

Since computing the wrong function could lead to harm, safety contracts must specify functional behaviour. This might be expressed in terms of pre- and post-conditions or valid operation sequences and in formal, informal, or mixed notation.

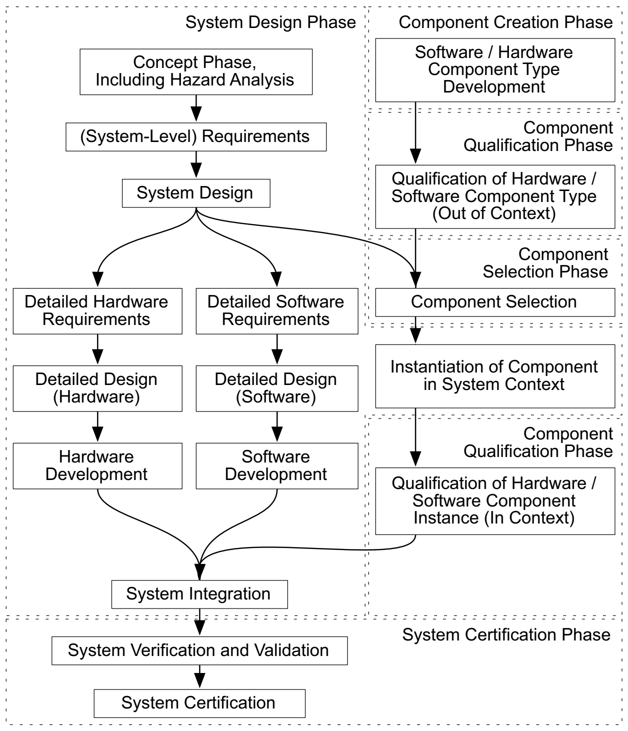
System Design Phase
Component Creation Phase
Concept Phase, Including Hazard Analysis
(System-Level) Requirements
System Design
Software / Hardware Component Type Development
Component Qualification Phase
Qualification of Hardware / Software Component Type (Out of Context)
Component Selection Phase
Component Selection
Detailed Hardware Requirements
Detailed Software Requirements
Instantiation of Component in System Context
Detailed Design (Hardware)
Detailed Design (Software)
Component Qualification Phase
Hardware Development
Software Development
Qualification of Hardware / Software Component Instance (In Context)
System Integration
System Certification Phase
System Verification and Validation
System Certification

Fig. 2. Phases of the Generic Process Model

1) Related Issues: When a component type is used in a context for which it was not expressly designed, behavioural aspects that its designers did not consider might be important. For example, the fact that a component temporarily suspends all interrupts might be unremarkable in some contexts and important in others. Moreover, some evidence of functional behaviour is system specific. For example, functional tests must use the chosen compiler and target computer if they are to reveal defects arising from compiler bugs or incorrect compiler settings. Developers instantiating a component type that is reused at the source-code level must re-execute functional tests.

2) Implications: Safety is a system-level property and a component type is not part of a system. We recommend not singling out component type contracts as safety contracts.

When a component type will be used on multiple platforms, its functional behaviour contracts indicate what what its instances are *expected* to do, not what evidence *shows* that they do. Contract and component creation mechanisms must make the need to generate system-specific evidence clear to developers. Where analyses of test plan requirements coverage and structural coverage are system-independent and can be qualified, component type contracts should guarantee related test plan properties. Component instance safety arguments can then refer to new test results and qualified test plan properties.

B. Timing Properties

Since late computation could also lead to harm, safety contracts must also specify timing properties such as (a) timing and timeliness of inputs and outputs, (b) component execution frequency, (c) Worst Case Execution Time (WCET), (d) interrupt handling and/or masking behaviour, and (e) synchronisation (e.g. assumption of single threading, use of a synchronisation primitive, or disabling of interrupts).

1) Related Issues: Compiling the same source code with a different compiler (even for the same platform) might result in object code differences that affect execution time. Even a component distributed as object code might run faster or slower if clock rates or cache settings differ. Achieving high utilisation often requires knowing WCET precisely [7]. But we cannot precisely know the execution time of a component type.

2) Implications: As with contracts for some resource usage properties, component type contracts for WCET might serve to *indicate* component instances' expected performance:

Assume:	X is compiled using GCC 4.2 with no optimisation, executed on a Freescale MPC 5554 clocked at 200$\pm$4 megahertz with all caches disabled, and not interrupted.
Guarantee	WCET(`X.foo()`) $\leq$ 1 millisecond.
Confidence	Informative only.

Component type WCET contracts might be the basis for component instance contracts that serve as placeholders for system design. However, typical WCET assessment techniques apply to entire tasks, not software components [7]. If WCET evidence must be collected anew each time a component is instantiated, it might make more sense to gather evidence about tasks' WCET than components' WCET.

IV. Conclusions

CBSE for safety-critical software systems is not as simple as it first appears. Because functional test evidence for source-code components must be regenerated, component type contracts should include qualifiable test plan properties instead. Because the meaning of execution time contracts changes during development, contracts must distinguish between indicative properties and properties backed by safety evidence (of a given quality). Further research is needed to ensure that SafeCer components and contracts are fit for purpose.

Acknowledgment

The work was funded by the Swedish Foundation for Strategic Research (SYNOPSIS) and Artemis (SafeCer).

References

[1] SafeCer. (2013, June) Safety certification of software-intensive systems with reusable components. [Online]. Available: http://www.safecer.eu

[2] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[3] A. Cimatti and S. Tonetta, "A property-based proof system for contract-based design," *Proc. 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 21–28, 2012.

[4] J. Carlson, C. Ekelin, J.-L. Gilbert, Á. Herranz, and S. Puri, "Generic component meta-model, Version 1.0," SafeCer, Deliverable D2.2.4, 2012.

[5] P. Graydon *et al.*, "Nature and derivation of safety contracts, Version 1.1," SafeCer, Deliverable D2.3.2, 2013.

[6] P. Böhm *et al.*, "Specification of the requirements on the generic component model, including certification properties and safety contracts," SafeCer, Deliverable D2.2.1 and D2.2.2, 2012.

[7] P. Graydon and I. Bate, "Realistic safety cases for the timing of systems," *The Computer Journal*, 2013, in press.