Zusicherung in der Anwendung von IEC 61508 Part 3

Prof. Peter Bernard Ladkin, Ph.D., Technische Fakultät und CITEC, Universität Bielefeld, und Causalis Limited, Bielefeld, Deutschland. ladkin@rvs.uni-bielefeld.de ladkin@causalis.com

Kurzfassung

Beschreibung von Entwicklungsverfahren und Dokumentation, die ermöglichen, sicher zu stellen, dass eine sicherheitsrelevante Software nach der Norm IEC 61508 gebrauchstauglich („fit for purpose“) ist.

1. Die Problematik

IEC 61508 Version 2 Teil 3 empfiehlt Methoden, die angewendet werden können oder sollen in der Entwicklung von Software (SW), denen eine bestimme SIL zugewiesen ist. SW bekäme eine SIL nach der zugelassenen Rate gefährlicher Fehler (dangerous failure rate) des (Sub)systems, das von der SW kontrolliert wird. Die HW-SIL-Fehlerraten, die in Teil 1 zugewiesen sind, sind laut wohlbekannten mathematischen Ergebnissen in der SW-Zuverlässigkeits-Statistik über die übliche SW-Test-Methodik nicht zu erreichen. Also ist die SW-Zusicherung primär über empfohlene Entwicklungsmethoden zu erreichen.

Problematisch hier ist, dass es keine beweisbare Verbindung gibt zwischen der Qualität des SW-Produktes und den Methoden an sich, die in der Entwicklung angewendet wurden. Die erfolgreiche Anwendung der empfohlenen Methoden ist sehr von der Erfahrung und der Sorgfalt des Entwicklers abhängig. Man kann nicht einfach sagen: wir haben dies-und-dies wie empfohlen angewandt und deshalb ist die Gefährliche-Fehler-Rate der SW so-und-so niedrig. Einige Firmen haben über Jahrzehnte ihre Methoden und den Erfolg und die daraus resultierende Qualität ihrer SW sorgfältig dokumentiert, um eine Basis von Fehlerratenachweisen zu erstellen; einige eben auch nicht (insbesondere kleinere Firmen, die noch nicht seit Jahrzehnten existieren!).

Der Autor kennt keine zuverlässige Messmethoden für ultrahoch-zuverlässige SW (SIL 1 oder höher) die allein von den angewandten Entwicklungsmethoden abhängig sind. Man kann aber objektive Eigenschaften des SW-Produktes mit der korrekten Anwendung einiger Überprüfungsmethoden erfolgreich feststellen und dies lässt die möglichen Fehlerquellen deutlich beschränken. Dieser Beitrag versucht zu zeigen, wie dies gemacht werden kann.

Es existiert eine Vielfalt von Methoden, meistens unter „Formale Methoden“ zusammengefasst, die zur Zusicherung einiger objektiven Eigneschaften der SW angewandt werden können, um einige Arten von Fehlern ganz auszuschliessen. Solche ausgereifte Methoden werden in Vorschlag G unter aufgelistet. Mit der Anwendung des Vorschlags G hat man also 26 Methoden, um mögliche Fehlerquellen in der SW zu bewältigen oder ganz auszuschliessen. Eine Pilotenstudie (SW für eine Art Zugleitbetrieb) hat gezeigt, dass es nicht aufwändig sein muss, den Vorschlag in den wesentlichen Punkten mit Hilfe kommerzieller Entwicklungstools zu erfüllen.

2. Teil der Lösung: Traceability (Nachvollziehbarkeit) in der Entwicklungskette

IEC 61508 beinhaltet Massnahmen, um die Traceability zwischen Safety Requirements Specification und Object Code sicherzustellen. Traceability bedeutet, dass ein Beweis vorliegt, dass der Object-Code die Safety Requirements Specification erfüllt. Es gibt hier einen Stand des Wissens / der Praxis (state of the art/state of the practice). Die hier dargestellte Folge von Vorschläge basieren auf diesem Stand.

Vorschlag A. Eine eindeutige, rigorose Functional Safety Requirements Specification (FRS) ist erforderlich. Die FRS muss die Überprüfung von (i) Konsistenz sowie (ii) relativer Vollständigkeit ermöglichen. Die Überprüfung muss gemacht werden und der Beweis im Safety Case vorliegen.

Die meisten Fehler in mittelkomplexen Systemen liegen in der Anpassung der FRS an die Realität der Benutzung begründet. Dies stellt also die erfolgversprechendste Stelle für die Einsatz der Fehlerbekämpfung dar.

Durch rigorose Verfahren wird versucht, sicherzustellen, dass die Fälle sorgfältig durchdacht sind. Konsistenz: Oft stellt sich heraus, dass die von verschiedenen Beteiligten gewünschte Funktionalität nicht gleichzeitig realisiert werden kann. Die kann an dieser Stelle durch Konsistenzprüfung erkannt werden, statt erst später während der Systementwicklung. Relative Vollständigkeit kann als Überbegriff angesehen werden, dafür zu sorgen, alle relevanten Szenarien abzudecken. Dies ist nicht üblich und erfordert formale Hilfsmittel.

Vorschlag B. (i) Die Software Architecture/Design Specification (SWA/DS) muss rigoros sein. (ii) Es muss ein formaler, rigoroser, korrekter Nachweis erbracht werden, dass die SWA/DS die FRS erfüllt.

Um sicherzustellen, dass das Design die Sicherheitsanforderungen erfüllt, hat sich gewisses Maß an formaler Strenge als notwendig erwiesen, besonders in der SWA/DS selbst.

Vorschlag C. (i) Falls die SW in einer Hochsprache, und nicht Maschinensprache, geschrieben ist, muss es eine

definierte Sprache geben, genannt "Executable Source Code Level" (ESCL); (ii) es muss streng formal, korrekt nachgewiesen werden, dass die SW im ESCL die SWA/DS erfüllt.

Software wird heute überwiegend in sogenannten "Hochsprachen" geschrieben; deren Sourcecode könnte die ESCL sein, beispielsweise für C, Modula, Ada oder SPARK. Für Java könnte die ESCL der Sourcecode oder der Bytecode sein. Für deklarative Sprachen wie Prolog ist die Definition einer ESCL nicht einfach. Der Vorschlag sieht jedoch deren Vorhandensein auf einer Stufe vor, normalerweise ist dies auch der Fall, da die meisten sicherheitskritische Systeme in prozeduralen Sprachen programmiert werden.

Die ESCL dient als sinnvolle Zwischenstufe zwischen SWA/DS und Objektcode. Es kann sinnvoll sein, viele solcher Zwischenstufen zu haben, aber es sollte mindestens eine geben.

Vorschlag D. *Definition:* Compilation ist als eine Operation definiert, die ESCL in Objektcode (OC) übersetzt, deren Bytes sich in der Hardware befinden. *Vorschlag*: Es muss einen rigorosen, korrekten Nachweis geben, dass der OC die ESCL erfüllt.

Dies ist der zweite Schritt von SWA/DS zum ausführbaren Code. Aber das ist noch nicht alles. Es gibt Grenzen des Hardwareeigenschaften, die nicht notwendigerweise aus der Bedeutung des Objektcodes hervorgehen. Daher:

Vorschlag E. Es muss einen rigorosen, korrekten Nachweis geben, dass Laufzeitfehler keine gefährlichen Fehler verursachen, oder zu ihnen beitragen.

Dies ist Stand der Technik, wird jedoch leider nicht immer angewendet. Dieser Vorschlag kann erfüllt werden, indem durch Verwendung von z. B. SPARK ganze Klassen von Laufzeitfehlern eliminiert werden, oder durch das Abfangen und Behandeln der Exceptions.

Vorschlagscluster F. Testen. Dieser Vorschlag wird hier nicht weiter spezifiziert.

Das Problem ist, dass es keine allgemeine Übereinstimmung darüber gibt, wie man testet, was man testen soll, und was die Tests zeigen. Es ist bekannt, dass statistisches Testen in der Praxis nicht das Erreichen eines SW-SIL auf dem Level der derzeit definierten HW-SILs zeigen kann. Andererseits ist jedoch klar, dass man durch routinemäßiges Testen eine gewisse Zusicherung der Eignung der Software für den vorgesehenen Verwendungszweck erhalten kann. Das schwierige daran - das Problem - ist es, was genau für eine Zusicherung man erhält, und zu spezifizieren, wie man diese erhält.

3. Wie die Vorschläge die Software-Qualität erhöhen

Die oben vorgestellten Vorschläge erlauben das Erreichen einer gewissen Qualität von Software durch das systematische Ausschließen bestimmer Fehlertypen.

Eine rigorose, nachgewiesen konsistente FRS vermeidet den Fehler, dass unterschiedliche Requirements sich gegenseitig in nicht offensichtlicher Weise widersprechen, und es unmöglich ist, ein System zu bauen, das die FRS erfüllt.

Eine rigorose, entsprechend einer Auffassung von Vollständigkeit bewiesen vollständige FRS vermeidet „Requirements-Fehler“, bei denen eine Betriebssituation entsteht, die nicht vorhergesehen wurde, und die nicht durch die Spezifikation abgedeckt ist, und in der sich das System unsicher verhält. Die Requirements-Fehler, die hier vermieden werden können, sind solche, die unter das verwendete Vollständigkeitskonzept fallen.

Ein rigoroser, korrekter Nachweise, dass die SWA/DS die FRS erfüllt, stellt sicher, dass das funktionale Design der SW eine der FRS entsprechendes Verhalten zeigt.

Ein rigoroser, korrekter Nachweiss, dass der ESCL-Code die SWA/DS erfüllt, zeigt, dass es keine Programmierfehler gibt.

Ein rigoroser, korrekter Nachweiss, dass der OC den ESCL-Code erfüllt, zeigt, dass es keine Compilationsfehler gibt.

Unter Verwendung dieser Vorschläge wird Software geliefert, die vollständig spezifiziert ist (entsprechend einer expliziten Auffassung von „Vollständigkeit"), und deren compilierter Code garantiert die Spezifikationen erfüllt (abgesehen von bestimmten, expliziten Formen von Laufzeitfehlern, die möglicherweise während der Entwicklung nicht ausgeschlossen wurden, deren Auswirkungen sich jedoch durch das Systemdesign mindern lassen.

Natürlich ist es möglich, dass die Garantien und Nachweise inkorrekt sind, aber der Grad der Korrektheit in Zusicherungsarbeiten dieser Art ist viel höher als der Grad der Korrektheit, der zur Zeit in der Industrie bei Code für sicherheitskritische Systeme erreicht wird (dieser Korrektheitsgrad wird derzeit auf etwa 1 bis 3 Fehler pro Tausend Zeilen Sourcecode, „KLOSC“ geschätzt.) Durch intensive, konsequente Anwendung von Zusicherungen während der Code-Entwicklung haben beispielsweise einige Firmen wiederholt eine Grad an Source-Code-Korrektheit nachgewiesen, der um ein bis zwei Größenordnungen höher als die erwähnten 1-3 Fehler pro KLOSC liegt. Dies ist der derzeitige Stand der Praxis.

4. Techniken und Empfehlungen

Es gibt einen gewisse Übereinstimmung unter international anerkannten Experten für sicherheitskritische Software, dass weitergehende Richtlinien zur Erfüllung der Vorschläge A bis E nötig sind, über die verschiedenen Anhänge und Tabellen hinaus. Die folgenden Methoden/Aufgaben/Produkte werden regelmäßig bei der Entwicklung von sicherer und hochzuverlässiger Software unterschiedlicher Art verwendet/erfüllt/erzeugt, es gibt Tools für sie und sie erlauben die Bewertung spezifischer Eigenschaften der resultierenden Software, wenn sie ordnungsgemäß angewendet wurden. Alle diese ausgereiften Methoden, Aufgaben und deren Produkte fallen in eine der folgenden Kategorien.

Vorschlag G. In der Dokumentation, die die Eignung für einen bestimmten Zweck belegt („Safety Case“) muss dargelegt werden, welche der folgenden Methoden benutzt/Aufgaben erfüllt/Produkte erzeugt wurden, und wie dies geleistet wurde.

1. Formale Functional Requirements Specification (FRS)
2. Formale Analyse der FRS
3. Formale Safety Requirements Specification (FSRS)
4. Formale Analyse der FSRS
5. Automatischer Beweis/Beweisüberprüfung („proof checking“) der Eigenschaften (Konsistenz, bestimmte Arten von Vollständigkeit) der FRS und FSRS
6. Formale Modellierung, model checking, und model exploration der FRS und FSRS

7. Formal Design Specification (FDS)
8. Formale Analyse der FDS
9. Automatischer Beweis/Beweisüberprüfung („proof checking“) der Erfüllung der FRS/ FSRS durch die FDS
10. Formale Modellierung, model checking, und model exploration der FDS
11. Formale deterministische statische Analyse der FDS (Informationsfluß, Datenflluß, Möglichkeit von Laufzeitfehlern)

12. Gemeinsame Entwicklung (codevelopment) von FDS und ESCL
13. Automatische Source-Code-Erzeugung aus FDS oder Zwischenspezifikation (intermediate specification, IS)
14. Automatischer Beweis/Beweisüberprüfung („proof checking“) der Erfüllung der FDS durch die IS

15. Automatische Erzeugung von Verifikationsbedingungen aus der ESCL
16. Rigorose Semantik der ESCL
17. Automatischer Beweis/Beweisüberprüfung („proof checking“) der Eigenschaften des ESCL-level (wie Anfälligkeit gegen bestimmte Arten von Laufzeitfehlern)
18. Automatischer Beweis/Beweisüberprüfung („proof checking“) der Erfüllung der FDS durch die ESCL

19. Formale Testerzeugung aus FRS
20. Formale Testerzeugung aus FSRS
21. Formale Testerzeugung aus FDS
22. Formale Testerzeugung aus IS
23. Formale Testerzeugung aus ESCL

24. Formale Analyse der Programmierstandards (SPARK, MISRA C, etc)
25. Analyse der Worst-Case Execution Time (WCET)

26. Monitor synthesis/runtime verification

Vorschlag H. Wenn während der Entwicklung bestimmte dieser 26 Elemente erfüllt/erzeugt wurden, so muss im Safety Case präzise gezeigt werden, welche Eigenschaften der Software hierdurch zugesichert werden.

Dies hat den Sinn, dem Entwickler und Gutachter gleichermaßen eine klare Vorstellung zu geben, welche Eigenschaften der Software durch welche Methoden zugesichert werden (und dass die Methoden sorgfältig und korrekt angewendet wurden.) Dies führt zu dem übergeordneten Ziel, dass:

Vorschlag J. Es muss im Safety Case demonstriert werden, dass der Objektcode die FRS erfüllt.

4.1 Beispiel

Im folgenden ein kleines hypothetisches Beispiel, wie ein Safety-Case-Eintrag zu Technik 9 aussehen könnte.

Technik 9, Automatischer Beweis/Beweisüberprüfung („proof checking“) der Erfüllung der FRS/ FSRS durch die FDS, wurde durchgeführt. Die verwendete Methode war TLA. Die FSRS ist in Z formuliert: Die FDS in TLA+. Daher wurde eine Übersetzung des Z in der FSRS nach TLA+ manuell durchgeführt (siehe Begleitdokumentation). Es wurde gezeigt, dass die resultierende TLA+-Spezifikation „machine-closed“ ist. TLA wurde manuell benutzt, um das Refinement der TLA+-Übersetzung der FSRS durch die FDS zu beweisen. Weil diese Aufgabe nur die FSRS betrifft, wurden das Refinement der Safety-Properties bewiesen. Das Merz-TLA-Prover für Isabelle wurde zum Proof-Checking des manuellen Beweises verwendet (siehe Begleitdokumentation), korrigiert (dto.) und die Korrekturen wurden ebenfalls mit dem Merz-Proof-Checker überprüft.

Literatur

Barnes, J., *High Integrity Software: The SPARK Approach to Safety and Security*, Addison-Wesley, 2003 & 2006.

Holzmann, G, *The SPIN Model Checker,* Addison-Wesley, 2004.

Ladkin, P. B., *Functional Safety of Software-Based Critical Systems*, Paper zum Keynote-Vortrag , The Ada Connection, Vereinigung der 16th International Conference on Reliable Software Technologies (Ada-Europe) und Ada-UK, Edinburgh, Juni 2011. Tagungsband wird erscheinen in der Reihe *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, 2011.

Paper verfügbar über
www.rvs.uni-bielefeld.de → Publications
sowie www.causalis.com → Publications

Ladkin, P. B., Sanders, J. und Sieker, B.M., *Computer Safety: Ein Lehrbuch,* Springer-Verlag, noch nicht erschienen. Auch verfügbar 2011 über
www.rvs.uni-bielefeld.de → Publications
sowie www.causalis.com → Publications

Danksagung

Der Autor dankt ganz herzlich Herrn Dr.-Ing. Bernd Sieker für Diskussion und Vorbereitungshilfe