Automatisierung von anforderungsbasiertem Testen

Ralf Gerlich, Rainer Gerlich
Dr. Rainer Gerlich BSSE System and Software Engineering (BSSE)

Manuelle *Anforderungstests* sind aufwändig: Die Eingabedaten müssen die Anforderungen abdecken, und beobachtete Ausgabedaten müssen auf ihre Kompatibilität mit den Anforderungen geprüft werden. Testfälle können auch automatisch aus Testmodellen erzeugt werden, die aber zunächst manuell erstellt werden müssen. Im Kontrast dazu nutzt der hier vorgestellte Ansatz einfachere Formen der Anforderungsformalisierung, um die Testdaten, die bei automatischen *Robustheitstests* mit *massiver Stimulation* erzeugt werden, auf Anforderungen abzubilden und die Ergebnisse auf Korrektheit zu prüfen.

Status der Verifikation von Anforderungen durch Funktionstests

Bild 1 zeigt die klassische Vorgehensweise der Verifikation von Anforderungen durch Funktionstests (Unit Tests) und manuell erstellte Testfälle. Auf der Basis von Anforderungen wird Quellcode erstellt und manuell eine Beziehung zwischen Anforderungen und Funktionen hergestellt. Aus den Anforderungen werden Testfälle für den Funktionstest abgeleitet. Eingabedaten und erwartete Ergebnisse werden in Testscripts überführt, die den Test ausführen und ihn als erfolgreich oder nicht (pass/fail) bewerten.

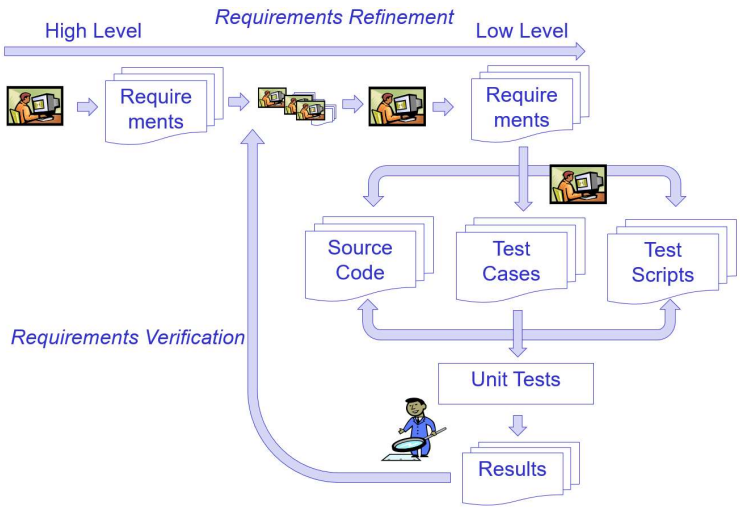

Bild 1: Verifikation durch manuell erstellte Testfälle

Modellgetriebene Ansätze für Anforderungstests

In den letzten Jahren wurden verschiedene Ansätze entwickelt, aus Modellen Anforderungstests abzuleiten. Dazu wird das gewünschte Verhalten modelliert und mit Annotationen zur Bestimmung der Anforderungsabdeckung und der Prüfung der Anforderungserfüllung versehen[1]. Durch Random Walks oder ähnliche Verfahren

werden dann Testsequenzen- und -daten aus diesen Modellen abgeleitet und manuell oder automatisch in Testskripte überführt.

Ein Spezifikationsmodell, das mit wenig Veränderung als Testmodell für eine solche Methode verwendet werden kann, ist ideal bei dieser Vorgehensweise. Ist dies nicht der Fall, muss zunächst ein solches Modell erstellt und mit der Spezifikation, die meistens nur in Textform vorliegt – manuell abgeglichen werden.

Automatisierte Robustheitstests

Automatisierte Robustheitstests – gelegentlich auch unter dem Begriff „Fuzzing“ zusammengefasst – werden verwendet, um die Robustheit einer Komponente gegen unerwünschte Eingaben zu prüfen[2]. Dazu wird eine Komponente etwa mit Zufallsdaten stimuliert, wobei auf Anomalien bei der Ausführung geachtet wird. So können teilweise auch funktionale Schwachstellen in der Komponente identifiziert werden. Die einfache Art der Testdatenerzeugung ermöglicht dabei einen großen Testdurchsatz – eine *massive* Stimulation.

Aus den Ergebnissen dieser Tests lassen sich jedoch keine Aussagen über Anforderungserfüllung oder -abdeckung ableiten. Eine Abbildung der Testeingaben auf die Anforderungen fehlt.

Automatisiertes anforderungsbasiertes Testen

Ziel des automatischen anforderungsbasierten Testen auf Codeebene ist es, eine Korrelation zwischen Anforderungen und Testfällen automatisiert herzustellen, und zugleich zu bestimmen, welche der Funktionen im Code jeweils durch welche Anforderung betroffen sind. Die Testdaten sollen automatisch erzeugt und angewendet werden, so dass sich der manuelle Aufwand drastisch reduziert.

Der gewählte Ansatz verwendet dabei drei Abbildungen[3]:

- von Eingabedaten zu Anforderungen (Anforderungsabdeckung),
- von Anforderungen auf Funktionen, und
- von Anforderungen zu *Orakeln*.

Dieses Prinzip ist in Bild 2 dargestellt. Maschinenlesbare Anforderungen werden in den Funktionstest integriert und während der massiven Stimulation ausgewertet. Somit ist es möglich, sowohl die Anforderungsabdeckung als auch die Erfüllung der Anforderungen für die ausgeführten Testfälle zu bestimmen. Bei Nichterfüllung beschreibt der zugehörige Testvektor aus Eingabe- und Ausgabedaten ein Gegenbeispiel.

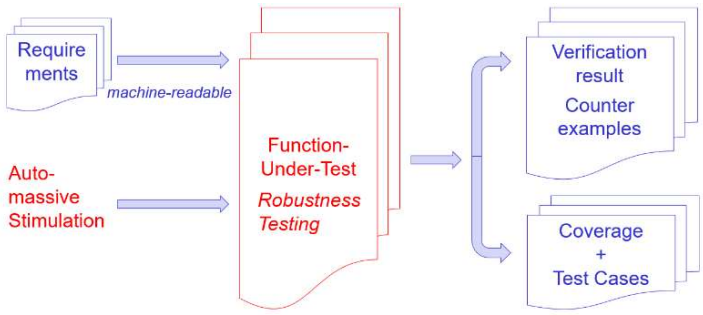

Bild 2: Prinzipielle Vorgehensweise

Letzteres geschieht über Orakel. Ein Orakel ist eine ausführbare Funktion, die für eine Eingabe und eine beobachtete Ausgabe ein *pass/fail/don't know*-Verdict erzeugt. Dabei ist es möglich, dass einzelne Orakel nur Entscheidungen für bestimmte echte Untermengen des Eingaberaums treffen. Idealerweise decken die Orakel gemeinsam aber den gesamten Eingaberaum ab.

Bild 3 zeigt weitere Details des automatischen Ablaufs. Durch die massive Stimulation wird der Eingaberaum über eine vorgebbare Anzahl von Testvektoren abgetastet. Auf jeden Testvektor wird das Orakel angewendet und das Ergebnis festgehalten. Auf diese Weise können Gegenbeispiele gefunden werden.

Zwischen Anforderungen und Funktionen besteht allgemein eine n:m-Beziehung. So können bestimmte Anforderungen, z.B. zur Rechengenauigkeit, auf mehrere Funktionen anzuwenden sein. Ebenso kann eine Funktion mehrere Anforderungen implementieren.

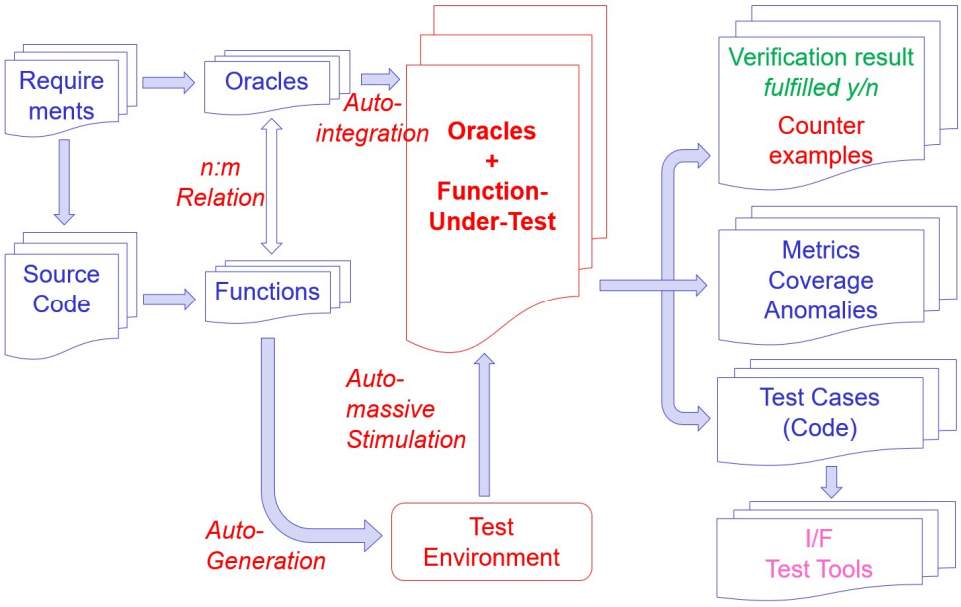

Bild 3: Details zum Ablauf

Um die Anforderungen und Funktionen einander zuzuordnen, müssen die betroffenen Elemente – Datenobjekte, Strukturen, Parameter – aus dem Text der Anforderung extrahiert und ihren Entsprechungen im Code zugeordnet werden. Dies kann durch Namensregeln geschehen, die Namen aus den Anforderungen in Namen im Code übersetzen oder direkt verwenden. So kann über die Beziehungen Orakel-Anforderung und Anforderung-Funktion eine Zuordnung von Orakeln zu den Funktionen erfolgen, auf die sie angewendet werden müssen. Eine aufwändige manuelle Zuordnung kann so vermieden werden.

In Bild 4 wird der logische Fluss im Kontext hierarchischer Anforderungen dargestellt. Top-down werden die Anforderungen detailliert bis zu einer Ebene, auf der sie schließlich in Code umgesetzt werden können. Erst auf dieser Ebene ist es sinnvoll, Orakel zu definieren. Anforderungen auf höheren Abstraktionsebenen sind üblicherweise nicht für die Verifikation durch Code-Level-Tests bzw. Unit Tests geeignet.

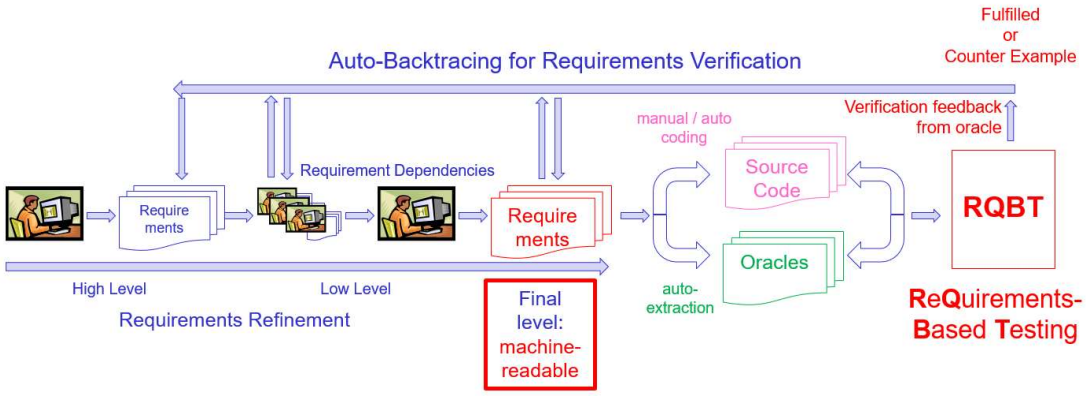

Bild 4: Gesamtablauf bei hierarchischen Anforderungen

Massive Stimulierung und Ableitung von Testvektoren

Für jede Funktion wird automatisch eine Testumgebung erstellt, über die die Funktion dann mit Daten aus dem Wertebereich der Funktionsparameter – inklusive globaler Variablen – stimuliert wird[2]. Dabei werden auch Spezialfälle wie z.B. -1 oder 0 für Ganzzahlparameter berücksichtigt. Auch andere Verfahren zur gezielten Abdeckung bestimmter Programmteile werden eingesetzt. Außerdem können invalide Werte eingespeist und Teile des Codes gezielt ergänzt werden, etwa um Speichermangel u.ä. zu simulieren.

Aus dem so betrachteten massiven Satz an Testeingaben und beobachteten Ausgaben werden interessante Fälle für die Erzeugung von Regressionstestsuites ausgewählt. Dies sind Eingaben, die zur Abdeckung beitragen, Ausnahmebedingungen auslösen oder die Vorgaben eines Orakels erfüllen oder verletzen. Diese Suites können dann auch mit externen Testmanagementwerkzeugen – z.B. Cantata oder VectorCAST – erneut ausgeführt und ihre Ergebnisse ausgewertet werden.

Weitere Analysen sind möglich: Eingabedaten, die zwar Code abdecken, aber keiner Anforderung bzw. Orakel zugeordnet werden können, deuten auf fehlende Anforderungen oder unnötigen Code hin. Wird für ein Orakel niemals die Vorbedingung im Eingaberaum aller zu testenden Funktionen erfüllt, handelt es sich um eine nicht abgedeckte Anforderung.

Der Orakel-Ansatz

Die Orakel in diesem Ansatz werden als temporale Implikationsbeziehungen dargestellt: *Wenn* vor dem Aufruf Bedingung A für die Eingabe gilt, *dann* muss nach dem Aufruf Bedingung B für Ein- und Ausgabe gelten (Bild 5). Ist Bedingung A nicht erfüllt, wird Bedingung B nicht ausgewertet und das Orakel liefert ein *don't know* als Ergebnis.

Es ist in dieser Struktur auch möglich, eine Tautologie – also einen immer wahren Ausdruck – als Vorbedingung zu nutzen. In diesem Fall muss die Bedingung B für jeden Testvektor erfüllt sein. Beispielsweise können so Anforderungen über Wertebereichsbeschränkungen der Ergebnisse umgesetzt werden.

Über bekannte Abhängigkeiten zwischen den Anforderungen kann das Ergebnis zu den Anforderungen auf höherer Ebene übertragen werden. Auf jeder Ebene kann somit festgestellt werden, welche Funktionen zu einem positiven oder negativen Ergebnis beitragen.

Die Verifikation über massive Stimulation basiert auf zwar sehr vielen, aber doch endlich vielen Testvektoren, d.h. eine vollständige Analyse des Zustandsraumes ist in der Regel nicht möglich. Jedoch übersteigt die Zahl der automatisch erzeugten Stimuli bei weitem die Zahl der manuell erzeugbaren Fälle, was insbesondere für das Finden von Gegenbeispielen relevant ist.

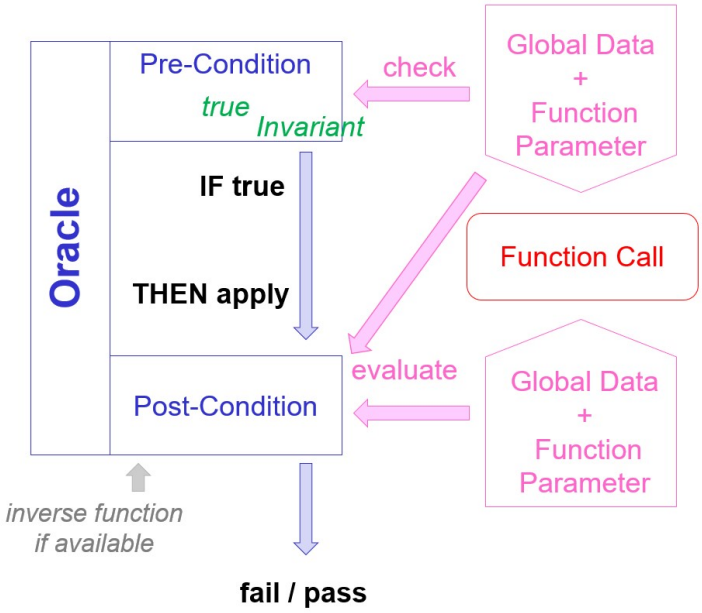

Bild 5: Orakel-Struktur

Beispiele

Betrachten wir als Beispiel ein Orakel für die Wurzelfunktion. Der einfache Ansatz $x \geq 0 \Rightarrow \sqrt{x}^2 = x$ wäre zwar mathematisch korrekt, wird aber bei endlicher numerischer Präzision nicht funktionieren. Korrekt wäre: $x \geq f_{min} \Rightarrow \left\| \frac{\sqrt{x}^2 - x}{x} \right\| < \varepsilon$, wobei f_{min} die kleinste normalisierte Fließkommazahl ist – nur für diese gilt die relative Fehlerabschätzung. Für $0 < x < f_{min}$ wären absolute Fehlergrenzen zu verwenden.

Noch einfacher erscheint die Definition eines Orakels für die *abs*-Funktion aus der C-Standard-Library: $x \in int \Rightarrow abs(x) \geq 0$. Interessanterweise ergibt sich dabei, dass der Wert von *abs(INT_MIN)* negativ ist, da *INT_MIN* selbst nicht in *int* darstellbar ist.

Auch sonst sind Typgrenzen zu beachten: Das Orakel $\left\| \frac{\sqrt{x^2} - x}{x} \right\| < \varepsilon$ für die Quadratfunktion wird viele scheinbare Gegenbeispiele produzieren, da für viele Werte von x die Quadrierung zum Überlauf führt.

Somit ergibt sich auch hier – wie bei vielen Ansätzen zur Formalisierung von Anforderungen – die Möglichkeit, Unzulänglichkeiten der Anforderungen zu erkennen.

Wahrheitstabellen, z.B. für Systemzustände, können sehr einfach dargestellt werden, wie Bild 6 zeigt.

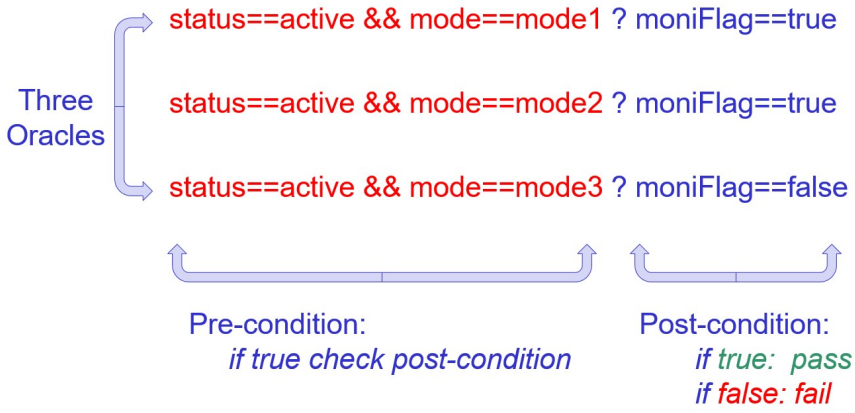

Bild 6: Überprüfung von Systemzuständen mit Orakeln

Notation der Anforderungen

Momentan werden die meisten Anforderungen als Freitext beschrieben, der nicht maschinell auswertbar ist. Eine Analyse solcher Anforderungen ergab, dass sie häufig unvollständig, nicht eindeutig oder inkonsistent und somit für dieses Verfahren ungeeignet sind. Bei der manuellen Verifikation müssen diese Schwächen dann durch Kreativität ausgeglichen werden.

Voraussetzung ist daher die Anwendung einer geeigneten Notation. Diese kann auch ein für den Anwender besser geeignetes Format als die Orakel-Form haben. Sie muss sich aber – automatisch – in die Orakel-Notation überführen lassen.

Anforderungen, die bereits in einem formalisierten Tabellenformat vorliegen, sind dafür gut geeignet.

Qualitätssicherung der Orakel

Wie jeder Code unterliegen auch Orakel der Qualitätssicherung. Fehler in den Orakeln können sonst dazu führen, dass Fehler in der Software übersehen werden. Da ein Orakel auf große Untermengen der Eingabemenge angewendet werden kann, ist es sogar möglich, dass Fehler mit großen Auswirkungen nicht erkannt werden, wenn das Orakel nicht korrekt ist. Dieses Risiko besteht aber auch bei der manuellen Erstellung von Testscripts oder Testcode. Dagegen können bei der automatischen Extraktion von Orakeln Maßnahmen zur Fehlervermeidung eingesetzt werden.

Ausblick
Da keine maschinen-lesbaren Anforderungen vorlagen, basiert die aktuelle Implementierung auf manuell in C-Code implementierten Orakeln, um die Machbarkeit und die Vorteile zu demonstrieren.
In zukünftigen Arbeiten soll eine abstraktere, aber für Anwender akzeptable Notation identifiziert werden. Dazu sollen textbasierte Anforderungen analysiert und in eine geeignete Notation umgesetzt werden, die automatisch in Orakel übersetzt werden kann. Dazu ist der enge Kontakt mit Anwendern erforderlich.

Danksagung

Der Inhalt dieses Beitrags ist ein Ergebnis des Projektes "Automated Source-code-based Testing, Continued" für die European Space Agency (ESA) (ESA Contract No. 4000116014) im Rahmen des General Support Technology Programme (GSTP). Das Budget wurde vom Bundesministeriums für Wirtschaft und Energie (BMWi) über das Raumfahrtmanagement des Deutschen Zentrum für Luft- und Raumfahrt bereitgestellt. Wir danken unserem Technical Officer, Maria Hernek (ESA), für ihre Unterstützung unserer Arbeiten und der Anregung, die massive Stimulation für anforderungsbasiertes Testen einzusetzen.

Literatur und Quellenverzeichnis
[1] H.-J. Herpel, G. Willich, J. Li, J. Xie, B. Johansen, K. Kvinnesland, S. Krueger, P. Barrios: "MATTS – A step towards Model Based Testing", Eurospace Symposium DASIA'2016 "DAta Systems in Aerospace", Mai 10-12, 2016, Tallinn, Estland

[2] R. Gerlich, R. Gerlich, M. Prochazka, K. Kvinnesland, B. Johansen: "A Case Study on Automated Source-Code-Based Testing Methods", Eurospace Symposium DASIA'2013 "DAta Systems in Aerospace", Mai 14-16, 2013, Porto, Portugal

[3] R. Gerlich, R. Gerlich, M. Hernek, J. Ramachandran, A. Pascoe, G. Johnson: "Challenges Regarding Automation of Requirements-based Testing", Eurospace Symposium DASIA'2017 "DAta Systems in Aerospace", Mai 30 – Juni 1, 2017, Göteborg, Schweden

Autoren:

Dr. rer. nat. Dipl.-Inf. Ralf Gerlich begann vor 20 Jahren mit der professionellen Softwareentwicklung im Raumfahrtumfeld, zunächst mit Schwerpunkt Systemprogrammierung. Inzwischen beschäftigt er sich hauptsächlich mit Methoden der Softwareverifikation in allen Ebenen der Softwareentwicklung (konstruktiv, organisatorisch und analytisch), sowohl manuell als auch werkzeuggestützt und vollautomatisch. Dabei gilt sein Interesse sowohl der theoretisch-mathematischen als auch der praktisch-pragmatischen Seite. Er ist seit Gründung der Firma Mitarbeiter bei „BSSE System and Software Engineering".

Dr. rer. nat. Dipl.-Phys. Rainer Gerlich verfügt über mehr als 35-jährige Erfahrung im industriellen Software Engineering, davon mehr als 30 Jahre in der Raumfahrt, mit Schwerpunkten in „Automation im Software-Lifecycle“, „Requirements Engineering“, „Verifikation“, „Automatisches Testen, „Embedded Systems“, „Sicherheitskritische Anwendungen“, „Projektmanagement“, „Werkzeugentwicklung“. Seit 1996 ist er Inhaber von „BSSE System and Software Engineering“.

Kontakt:

Ralf.Gerlich@bsse.biz

Rainer.Gerlich@bsse.biz