

05/2008

Software-Testmethoden

Was den Code stark macht

Wenn die Software ernsthafte Probleme bereitet, liegt dies meist an einer verspäteten Fehlersuche. Moderne Prüfmethode wie die statische Verifikation leisten in allen Phasen Unterstützung – vom Design bis zur Validierung. Sie garantieren robusten Code, auch wenn der Zeitrahmen für das Projekt eng gesteckt ist.

Wer die Qualität von Steuerungssoftware sichern will, darf sich nicht nur auf ihre Funktionen konzentrieren. Diese benötigen robusten Code als festes Fundament, auf dem sie realisiert werden können. Dies gilt für alle Arten von Produkten – von der Kaffeemaschine bis zum Hightech-Automobil.

Als eine grundlegende Anforderung an den Code gilt, dass er für gültige Eingaben gültige Ausgabewerte liefert. Schwieriger wird es, wenn die Software ungültige Eingangswerte ablehnen soll – eine weitere Bedingung für starken Code. Außerdem vermeidet robuste Software Abstürze und führt keine unbeabsichtigten Operationen aus.

Letztlich äußert sich jedes Code-Problem durch Fehler im Laufzeitverhalten, die unterschiedliche Formen annehmen. Beispielsweise reagiert Software prompt auf Divisionen durch Null, Wurzeln aus negativen Zahlen oder eine fehlerhafte Pointer-Arithmetik: Sie stürzt sofort ab. Verspätet antwortet das System dagegen meist auf nicht initialisierte Variablen. Auch die Überschreitung von Feldgrenzen macht sich oft erst langfristig, aber dann spontan unangenehm bemerkbar. Das Programm verabschiedet sich nämlich dann, wenn es zufälligerweise eine Variable am Ende des Feldes verwendet, die nicht mehr funktioniert.

Ein weiteres nicht zu unterschätzendes Problem sind Unter- und Überläufe bei arithmetischen Operationen. Hier besteht wiederum ein altbekanntes Problem: Aufgrund der Zeitverzögerung sind zwischen einem Fehlverhalten und seiner Ursache nur schwer Verbindungen herstellbar.

In 80% der Projekte kümmern sich Teams zu spät um Fehler

Trotz aller Bemühungen lassen sich Fehler nicht komplett vermeiden. Sie gehören zum Programmieralltag. Allerdings wird schwacher Code zum ernsthaften Problem, wenn der Entwickler ihn nicht rechtzeitig verbessert. Das böse Erwachen kommt oft erst im Feldeinsatz oder bei der nächsten Version. Der Entwickler trägt die Hauptverantwortung für die Qualität des Endprodukts, weil er das Innenleben des Programms am besten kennt. Deshalb muss er von Entwicklungsbeginn an auf die Robustheit seines Codes achten.

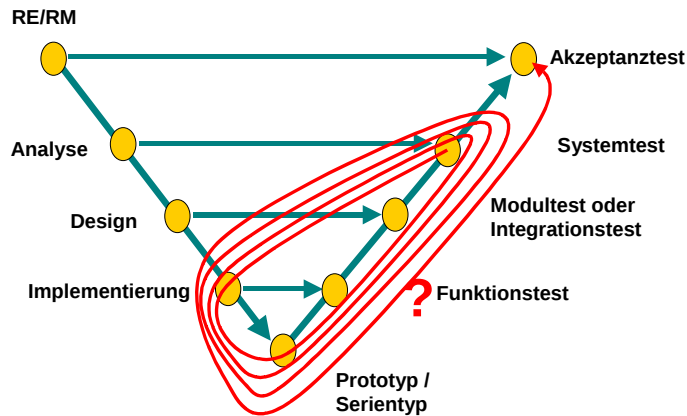


Bild 1: Viele Fehler führen zu vielen Schleifen zwischen Systemtest und Implementierung. Hoher Zeitverlust ist die Folge.

Die Praxis sieht heute allerdings anders aus, denn bei 80% aller Projekte kümmert sich das Team zu spät um Softwarezeitbomben. Aufgrund des hohen Zeitdrucks verzichtet es beispielsweise ganz oder teilweise auf Unit- bzw. Funktionstests. Dies führt in späteren Phasen zu mangelhafter Robustheit des Gesamtsystems und damit zu vielen Schleifen zwischen Systemtest und Implementierung. Trotz des Mehraufwands leidet die Qualität, denn einige Fehler haben sich im Laufe der Zeit fest eingenistet. Sie werden später den Kunden verärgern.

Eine frühzeitige Code-Prüfung vermeidet also nicht nur unnötige Iterationen und verkürzt die Testphase spürbar, sondern nimmt auch direkt Einfluss auf den Verkaufserfolg. Zwei erprobte Methoden können Entwickler dabei anwenden: die statische Prüfung und den dynamischen Test. In die erste Kategorie fallen manuelle Reviews, bei denen das Team unter anderem den Code mit den Dokumenten abgleicht. Die Basis können Metriken zur Code-Komplexität bilden, außerdem Programmierrichtlinien, mit denen sich fehlerträchtige Konstruktionen vermeiden lassen.

Diese verbindlichen Regeln kann jedes Unternehmen selbst aufstellen, oder sich an einen Standard halten, wie beispielsweise MISRA, der seine Ursprünge im Automobilbereich hat. Die automatische Prüfung des Codes auf diesen Standard hin übernehmen Software-Tools wie PC-Lint von Gimpel Software, QA-C MISRA von QA Systems oder der PolySpace MISRA Checker. Auch Compiler wie der C167 von Tasking integrieren bereits eine MISRA-Prüfung.

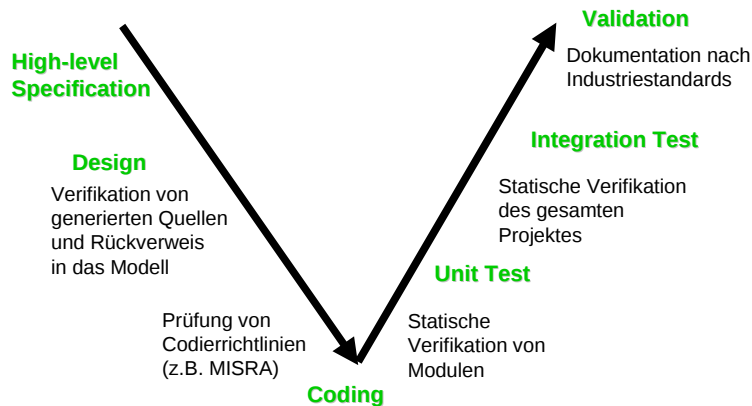


Bild 2: Die statische Verifikation kann jede Projektphase vom Entwurf bis zur Validierung unterstützen.

Neben der statischen Prüfung leistet der dynamische Test bereits in frühen Projektphasen als Unit- bzw. Funktionstest gute Arbeit. Wer hier besonders hohe Ansprüche stellt, stößt mit herkömmlichen Methoden allerdings schnell an seine Grenzen. Schließlich sollte alles, was eine Funktion in irgendeiner Form beeinflussen kann, durch Testfälle abgedeckt und reproduzierbar sein. Dies betrifft alle Kombinationen von Daten und Pfaden. Ein unmögliches Unterfangen, denn die Zahl der Testfälle geht dann gegen unendlich.

Die richtigen Testfälle wählen

Weil Vollständigkeit unmöglich realisierbar ist, muss sich das Projektteam auf wenige, aussagekräftige Tests konzentrieren, die es unter anderem durch Äquivalenzklassenbildung und Grenzwertanalysen ermitteln kann. Dynamische Tests lassen sich auch durch statische Prüfungen ausgezeichnet vorbereiten, weil sie die Selektion erleichtern. Manuelle Methoden sind dabei immens zeit- und kostenaufwändig, aber hocheffektiv.

Durch gut geplante und ausgeführte Reviews lassen sich zwischen 60 und 90% aller Fehler in Spezifikation und Code aufspüren. Der Nachteil: In der Praxis schaffen vier Prüfer in der Stunde nur zwischen 60 und hundert Lines of Code (LOC). Reviews sind also eher bei schriftlichen Dokumenten erste Wahl, die sich nicht testen lassen. Außerdem lässt sich die Prüfung nicht reproduzieren, weil sie vom Wissensstand und der Tagesform des Teams abhängt.

Im Code-Bereich geht der Trend eindeutig in Richtung automatisierter Verfahren, denn Werkzeuge wie LINT oder PolySpace bringen konstante Leistung. Sie können dieselbe Prüfkombination beliebig oft wiederholen und erfordern zudem vergleichsweise wenig Arbeitsaufwand. Letztlich setzen Tester nur die Analyse auf und begutachten nach dem automatischen Durchlauf die Ergebnisse. Allerdings müssen Unternehmen genau kalkulieren, ob sich die Investition in Lizenzen langfristig rechnet oder eine manuelle Prüfung aus Kostengründen vorzuziehen ist.

Erst mathematische Verfahren liefern eindeutige Beweise für Fehlerfreiheit

Wie auch immer sich das Management entscheidet, einen Haken besitzen beide Methoden: Sie können die Korrektheit des Systems nicht beweisen. Wenn die Prüfer an einer Stelle keinen Fehler finden, bedeutet dies nicht, dass sich dort keiner versteckt hält. Erst die statische Verifikation auf Basis mathematischer Verfahren liefert dafür den eindeutigen Beweis. Diesen neuen Weg beschreitet das Tool PolySpace.

Seine integrierten Algorithmen finden alle Fehler, wie sie auch durch herkömmliche Methoden aufgespürt werden, nur arbeitet die Software wesentlich schneller und sorgfältiger. Gleichzeitig spürt sie aber auch bisher nicht erkennbare Fehler auf, weil sie im gleichen Zeitraum wesentlich mehr Testszenarien durchspielen kann. Dafür hebt PolySpace den Code, vergleichbar der Laplace- oder Fourier-Transformation, auf eine andere Ebene. Dort lassen sich komplexe Vorgänge einfacher rechnen.

Die statische Verifikation perfektioniert die Fehlersuche

Die statische Verifikation testet jeden Pfad in jeder Kombination. Gleichzeitig prüft sie alle Eingangswerte in jeder Variation durch alle Pfade im System. Damit ist klar, warum diese Methode den klassischen White-Box-Test weitgehend ersetzt: Dieser beschränkt sich auf eine der beiden Dimensionen, auf die der Entwickler den Code vorbereitet hat. Die statische Verifikation testet und bewertet Strukturen sowie Daten dagegen in einem Durchlauf. Dies fast ausschließlich auf dem Quellcode, so dass das Projektteam keine Testfälle und -treiber mehr erstellen muss.

Das Analyseergebnis vermittelt PolySpace anschaulich im Code. Bei einer roten Markierung tritt immer ein Laufzeitfehler auf, unabhängig vom Zweig, der Call-Beziehung oder den verwendeten Daten. Bei Orange ist in Einzelfällen mit einem Fehler zu rechnen und grauer Code ist tot, wird also nicht ausgeführt. Tester bewerten die Bedeutung der einzelnen Farben im Programmkontext und leiten daraus den weiteren Handlungsbedarf ab. Bei grünem Code können sie sich entspannt zurücklehnen, weil dessen Korrektheit mathematisch bewiesen ist.

Die statische Verifikation lässt sich auch in der modellbasierten Entwicklung einsetzen. Es gibt z.B. Anbindungen an die Tools Rhapsody von I-Logix und Matlab von Mathworks. Hier wird der Code unmittelbar verifiziert, nachdem die Software ihn erstellt hat. Laufzeitfehler lassen dabei Rückschlüsse auf Inkonsistenzen im Modell zu.

Effiziente Tools und erfahrene Entwickler sind Garant für hohe Softwarequalität

Einen weiteren Einsatzbereich der statischen Verifikation bildet die manuelle Programmierung. Hier können Tester den Code anhand von Richtlinien wie MISRA auf den Prüfstand stellen. Außerdem kann der Programmierer im Unit-Test Funktion für Funktion, und beim Integrationstest das gesamte System verifizieren. Im Bereich der Validierung profitieren Unternehmen vor allem dann von der statischen Verifikation, wenn sie in Zertifizierungsaktivitäten eingebunden sind. Dies ist meist bei sicherheitsrelevanten Systemen der Fall. Das Projektteam leitet aus den Verifikationsergebnissen Dokumentationen ab, die den geforderten Industriestandards entsprechen.

So effizient ein Tool wie PolySpace auf allen Projektebenen seine Dienste verrichten mag, ist es doch ohne erfahrene Entwickler und Tester nur die Hälfte wert. Wer beispielsweise bei der Modularisierung in der Designphase schlampig arbeitet, baut früh zahlreiche Fehlerquellen ein. Die statische Verifikation kann diese zwar aufspüren, aber nicht in ihrer Tragweite bewerten und schon gar nicht eliminieren. Deshalb bewahrheitet sich auch hier wieder: Je besser ein Projektteam arbeitet, desto mehr profitiert es von seinen Programmier-Tools.

Autor:

Dipl.-Ing. Frank Listing ist Trainer und Projektcoach bei MicroConsult. Zuvor arbeitete er zehn Jahre in der Software-Entwicklung. Seine Schwerpunkte sind Microsoft-Plattformen, objektorientierte Programmierung sowie Software-Test.
Kontakt: f.listing@microconsult.com, . Tel. +49 89 450617-0

MicroConsult GmbH:

Training, Coaching und Engineering für Software- und Hardwareentwicklung in der Industrie.
www.microconsult.de
Charles-de-Gaulle-Str. 6, 81737 München