

Wirksame Mittel gegen Software-Kinderkrankheiten

»Wir müssen Softwareprobleme von vorneherein beheben«

Verteilte Entwicklung und immer kürzer werdende Entwicklungszyklen erschweren das Leben von Softwareentwicklern. Bereits kleine Fehler können zu schweren Defekten führen. Mit seinen Tools will Coverity deshalb viele Fehler schon sehr früh identifizieren und beseitigen.

Man erinnere sich nur an das Chaos auf dem Londoner Flughafen Heathrow im Februar 2008: Ein misslungenes Software-Update hatte das Gepäcksystem lahm gelegt. Das tagelang andauernde Wirrwarr führte zum Ausfall zahlreicher Flüge und brachte der British Airways neben einer großen Blamage Kosten in Höhe von 20 Mio. Euro ein. Software wird immer komplexer, ausgeklügelter und steckt mittlerweile in fast allen Geräten – vom Telefon bis zum Herzschrittmacher. Die möglichen Konsequenzen eines Ausfalls sind dramatisch und zeigen, wie wichtig es ist, die Fehler einer Software im Entwicklungsstadium zu beheben. »71 Prozent der Entwicklungsprojekte verzögern sich, überschreiten das Budget oder verlaufen ganz im Sand. Viel Zeit und Geld werden in Fehlerbehebung und nachträgliche Software-

optimierung investiert. Aber nur ein Zehntel des Aufwands fließt in Bemühungen, Software sofort richtig zu entwickeln. Doch genau dadurch könnte man viele Pannen vermeiden und bares Geld sparen«, ist sich Ben Chelf, Mitbegründer und CTO von Coverity, sicher. »Der Code ist ein effizienter Ansatzpunkt zur Softwareoptimierung. Mit der Analyse und Visualisierung lassen sich Sicherheits- und Qualitätsprobleme früh identifizieren und beseitigen.«

Coverity hat sich deshalb auf Tools zur frühzeitigen Optimierung von Software spezialisiert. Die Grundlage bildet das patentierte »Software DNA Map«-Analysesystem. Dahinter verbirgt sich eine Bit-genaue äquivalente Darstellung des gesamten Softwaresystems, die automatisch erstellt wird. Sie lässt sich für verschiedene Analysen einsetzen, etwa für

die IT-Architektur, den statischen Code und die Standards sowie für die Komplexität. Sie enthält nicht nur die Syntax jeder Quelldatei im Softwaresystem, sondern versteht auch, wie die Software in ein finales Set an Objekt-, Exe- und Jar-Dateien eingefügt wird. Automatische Analysetechnologien, die den ganzen Code und seinen Aufbau verstehen, haben eine ganzheitliche, umfassende Sicht der Welt, die sie analysieren. »Die Software DNA Map enthält Quellcode Frontends und Bytecode-Readers. Man kann sie mit der Arbeit eines Compilers vergleichen, der Code in das gewünschte Format bringt. Wir speichern einfach das Analyseergebnis des Frontends in einer syntaktisch abstrakten Baumstruktur, die für die Analyse besser geeignet ist als Byte Code, CLR oder Objektdateien.«

Wichtiger Aspekt im Zusammenhang mit der Software DNA Map ist deren automatische Erstellung. Schließlich gibt es eine Unmenge an Informationen über Quelldateien und darüber, wie sie assembliert werden. »Ein Unternehmen zu bitten, diese Informationen manuell zu sammeln, um sie dann automatisch zu analysieren, würde der Idee der Automatisierung widersprechen. Deswegen muss es möglich sein, die Software DNA Map automatisch zu generieren.« Dies funktioniert mittels einer Technologie, die jede native Build-Umgebung und ihre Operationen betrachtet. »Beobachtet man das Zusammenspiel des Builds mit dem Betriebssystem, kann man nicht nur den gesamten Software-Assemblierungsprozess automatisch speichern, sondern man kann auch sehen, wie jede Datei übersetzt ist, und sie dann ein zweites Mal mit den besagten Frontends übersetzen.«

Setzt man das Analysetool während der Testphase ein, hat man die Chance, viele Fehler zu erkennen und zu beheben, bevor



Ben Chelf, Coverity

» Softwareentwicklungsunternehmen müssen aufhören, Softwareprobleme nachträglich zu behandeln. Sondern sie müssen Softwareprobleme von vorneherein vermeiden und ihre Produkte richtig bauen. «

die Software auf den Markt kommt. Es ist allerdings unmöglich, alle erdenklichen Szenarien durchzuspielen, weshalb Chelf rät, sich auf die wichtigsten, fehleranfälligen Stellen im Code zu fokussieren. »Unser Architecture Analyzer macht die komplexen Stellen des Systems sichtbar – also die Stellen, wo am ehesten Probleme auftreten.« Mit den automatisch erstellten Diagrammen und Strukturmatrizen der Abhängigkeiten liefert der Architecture Analyzer verwertbare Daten zum besseren Verständnis der Codestruktur. Auf Grundlage dieser Daten lässt sich auch ermitteln, ob Änderungen nach der Kompilierung den Softwareentwurf widerspiegeln und im Einklang mit den ursprünglichen Entwicklungsvorgaben stehen. Indem Softwarearchitekten eine Soll-Codearchitektur festlegen und die Codestruktur im Verlauf der Entwicklung des Quellcodes analysieren, können sie die Vorgaben besser durchsetzen. Auch potenzielle Defekte und architekturbedingte Sicherheitsbedrohungen, wie etwa mögliche Umgehungen der Kontrollpunkte für die Zugriffsteuerung auf eine Anwendung oder auf Verschlüsselungs- und Entschlüsselungs-APIs, erkennt das Programm automatisch und beugt somit Sicherheitslücken vor.



Architektur-Analysator

Der »Coverity Architecture Analyzer« visualisiert automatisch die architektonische Struktur und die Abhängigkeiten in großen, komplexen Codebasen. Er setzt auf dem »Coverity Software DNA Map«-Analysesystem auf und verhilft zu einem besseren Verständnis des kompletten Codes. Die Vorteile auf einen Blick:

- Identifikation komplexer Bereiche, die schwierig zu ändern und anfällig für Fehler sind.
- Festlegung und Durchsetzung von Design-Anforderungen, um ungewollte Abhängigkeiten während des Entwicklungsprozesses zu verhindern.
- Ermittlung des Ausmaßes der Änderungen vor deren Ausführung.
- Macht Codebasen begreiflich, die nicht dokumentiert wurden.
- Reduzierung der Gesamtbetriebskosten durch höhere Wartungsfreundlichkeit und weniger Fehler.

Der Architecture Analyzer stellt kritische Informationen bereit, etwa ob die gegenwärtige Struktur mit der Referenz-Struktur übereinstimmt. Aufgrund seines Web-basierten Interfaces und dem IDE-Plugin für Java erhält das gesamte Entwicklungsteam die benötigten Informationen zur richtigen Zeit. Das erleichtert eine kontrollierte Entwicklung großer Codebasen. (mk)

Eine der Herausforderungen bei der Softwareentwicklung ist die Einhaltung der ursprünglichen Design-Spezifikationen bei Modifikationen und Versionierungen. Das größte Problem dabei ist, das Design auf dem System durchzusetzen, während es geschrieben und umgewandelt wird. »Architekten entwickeln großartige Designs, aber kommunizieren diese mit Whiteboards, UML-Diagrammen und Word-Dokumenten. Wenn ein Bauarbeiter sich den Entwurf eines Gebäudes ansieht und die Anweisungen nicht befolgt, dann würde man ihn feuern. Wenn ein Entwickler sich einen Entwurf für ein Softwaresystem anschaut und den Anweisungen nicht folgt, dann ist es gut möglich, dass es niemandem auffällt.«

Auch die Designqualität lässt sich überprüfen, weiß Chelf: »Einige schlechte Design-Entscheidungen können sehr leicht mit dem Architecture Analyzer entdeckt werden. So ist etwa ein Code

mit Unmengen an zyklischen Abhängigkeiten von Grund auf eine schlechte Idee. Das erkennt das Programm auch sofort. Es gibt jedoch andere Entscheidungen, die gut oder schlecht sein können, aber kaum zu erkennen sind. Einen n^2 -Logarithmus über einen $n(\log n)$ -Logarithmus zu wählen, wenn n groß ist, wäre dumm. Eine ähnlich schlechte Entscheidung wäre die Wahl eines Designs, das zwar einige Zyklen einspart, aber schwieriger zu implementieren ist und somit eher zu Bugs neigt. Diese Dinge geschehen auch trotz Automatisierung.«

Der Architecture Analyzer ist kein Design-Pattern per se, sondern er beschäftigt sich hauptsächlich mit den Abhängigkeiten innerhalb eines Softwaresystems. Er unterstützt den Softwarearchitekten bei der Entwicklung einer geschichteten Architektur und verhindert zyklische Abhängigkeiten. Wie in der Literatur beschrieben, sollte man bei einer automa-

tisierten Analyse von Design-Pattern mit einer ersten Darstellung des Softwaresystems beginnen, was die Software DNA Map ermöglicht. Dann sollte man ein Bild aller Abhängigkeiten im Softwaresystem kreieren, wofür sich der Architecture Analyzer eignet.

»Wir sehen in den nächsten Monaten einen verstärkten Trend zu einer 'agilen Softwareentwicklung', die auf mehr Flexibilität und eine Verschlinkung des Entwicklungsprozesses setzt.« Dieser Ansatz erfordert ausgeklügelte Produkte, die Probleme frühzeitig und automatisch erkennen. Damit kommen Unternehmen schnell in den Genuss verkürzter Feedbackschleifen. So plädiert auch Chelf für eine rechtzeitige Qualitätskontrolle: »Softwareentwicklungsunternehmen müssen aufhören, Softwareprobleme nachträglich zu behandeln. Sondern sie müssen Softwareprobleme von vorneherein vermeiden und ihre Produkt richtig bauen.« Die automatisier-

te Analyse fördert diese Denkweise. Schon in naher Zukunft wird man gründlichere und schnellere Analysen in den ersten Phasen des Entwicklungsprozesses im Einsatz sehen. »Das Analysewerkzeug wird förmlich als 'automatisierter Programmierpartner' auf dem Schreibtisch des Entwicklers sitzen und ihm zuflüstern 'Hast du diesen Fall bedacht?', 'Hier ist ein Bug!' oder 'Du schadest dem Design des Systems!'. In den meisten Fällen wird es sich als richtig erweisen.« Laut Chelfs Einschätzung werden auch Berechnungen für Builds, Tests und die automatisierte Analyse über multiple Kerne und Maschinen aufgegliedert, um die gesamte verfügbare Rechenkraft auszuschöpfen.

Erweist sich die nächste Generation an Softwareentwicklungsprogrammen als erfolgreich, können Entwicklungsunternehmen ihre Produkte schneller und mit höherer Qualität auf den Markt bringen. (mk) ■