Embedded-Anbieter im Dilemma zwischen Marktdruck und Verantwortung: Wie können Software-Testwerkzeuge helfen?

Von Klaus Lambertz (Geschäftsführer Verifysoft Technology GmbH)
und Royd Lüdtke (Direktor Statische Codeanalyse bei Verifysoft Technology GmbH)

Bei der Entwicklung von Embedded Devices und Embedded Software gilt wie in vielen anderen Projekten das Modell des magischen Dreiecks, in dem Sachziele bzw. Qualität mit Kosten und Zeit in Beziehung gesetzt werden.

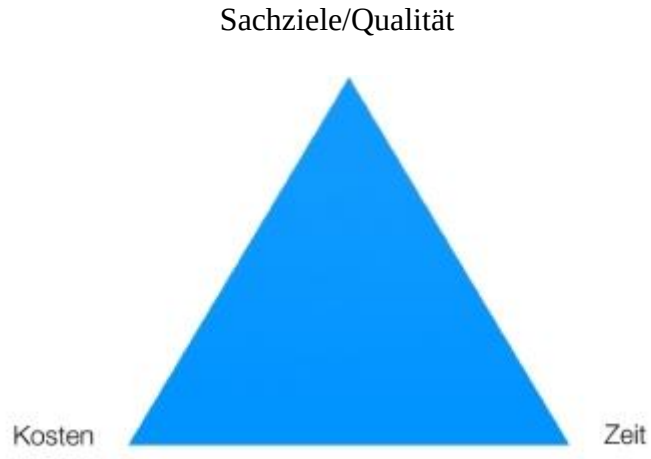

Abb. 1: Das magische Dreieck der Projektentwicklung: Qualität, Kosten und Zeit stehen in Beziehung zueinander.

Vor dem Entwicklungsstart werden die drei Größen definiert und priorisiert. Es wird festgelegt, welche Sachziele man bis zu welchen Terminen mit welchem Budget umsetzen will. Wird später eine der drei Größen verändert, hat das direkte Auswirkungen auf die beiden anderen Größen.

Bei der Entwicklung von Embedded Software können vor allem die Qualität und die Produktivität (Zeit) durch geeignete Software(test)-Werkzeuge positiv beeinflusst werden.

“Testen ist ökonomisch sinnvoll, solange die Kosten für das Finden und Beseitigen eines Fehlers im Test niedriger sind als die Kosten, die mit dem Auftreten eines Fehlers bei der Nutzung verbunden sind.” Quelle: Pol, M.; Koomen, T.; Spillner, A.: Management und Optimierung des Testprozesses, 2. Auflage, dpunkt, 2002

Diese Aussage wird jedem einleuchten, ist aber dennoch schwierig zu fassen. Während Kosten für Testsoftware und die Arbeitszeit der Tester einigermaßen transparent sind, ist der Nutzen einer besseren Reputation, Reduzierung der Entwicklungszeit, geringerer Wartungsaufwand und bessere Wiederverwendbarkeit der Softwaremodule im Voraus meist nicht in Euro und Cent zu beziffern. Das mag der Grund dafür sein, dass Softwaretestwerkzeuge trotz zahlreicher Vorteile noch nicht überall ausreichend genutzt werden. Ein großes Potential zur Beschleunigung der Softwareentwicklung und zur Steigerung der Qualität bleibt somit ungenutzt.

Testen früh im Entwickungsprozess mit Statischer Codeanalyse

In der Analytischen Qualitätssicherung unterscheidet man grundsätzlich zwischen statischen und dynamischen Testverfahren. Während bei der dynamischen Analyse der Code ausgeführt werden muss, ist dies bei der statischen Analyse nicht der Fall. Daher können statische Analysetools schon früh im Entwicklungsprozess in der Implementierungs-Phase genutzt werden und tragen deshalb massiv zum Projekterfolg bei, denn je früher getestet wird, desto günstiger die Fehlerbehebung.

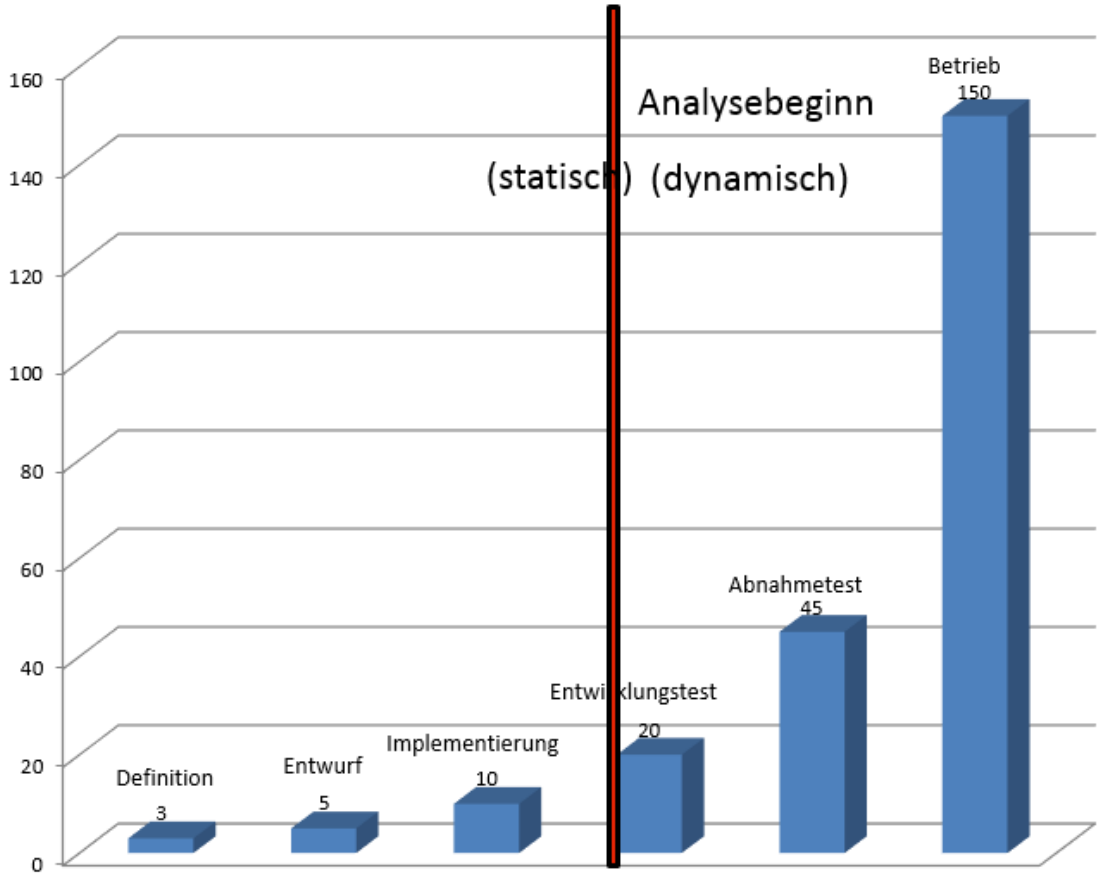

Abb. 2: Relative Kosten der Fehlerbehebung nach Barry W. Boehm: Software Engineering Economics: Statische Codeanalysen sind schon während der Implementierung möglich und sparen somit überproportional Kosten ein.

.

Statische Codeanalysetools überprüfen den Code auf Syntax, Semantik, Kontrollfluss- und Datenflussanomalien, Nebenläufigkeitsprobleme und Einhaltung von Codier-Richtlinien/Standards. Werkzeuge für die statische Analyse decken ohne Schreiben von Testfällen eine Vielzahl von Bugs auf und identifizieren sicherheitsgefährdende Schwachstellen. Ebenfalls erheben statische Analysetools Code-Metriken. Diese Maßzahlen geben einen Hinweis auf die sogenannte innere Qualität von Software. Hierzu gehören Wartbarkeit, Erweiterbarkeit, Analysierbarkeit und Testbarkeit der Software. Um die Softwarequalität langfristig zu sichern fordern zahlreiche Käufer von Software den Nachweis des Erreichens von bestimmten Metrik-Schwellwerten.

Idealerweise wird der Code bereits vom Beginn der Entwicklung an regelmäßig – am besten bereits durch den einzelnen Entwickler vor dem Einchecken und dann mit zunehmender Integration von Modulen - statisch untersucht. Hierdurch dürfte die Anzahl der Fehlermeldungen bei der Abschlussprüfung vor Auslieferung relativ überschaubar sein.

Sinnvoll ist es, den Code erst dann einem weiteren Verifikationsschritt wie Code-Reviews, Unit-Tests oder Integrationstests zuzuführen, wenn die Statische Codeanalyse keine Fehler mehr anzeigt.

Wenn man vergleicht, wie schnell potenzielle Probleme durch statische Analysetools aufgedeckt werden, die durch späteres Debuggen – wenn überhaupt – nur mit sehr viel Zeitaufwand aufzufinden sind, wird man schnell vom Einsatz statischer Codeanalyse-Werkzeuge überzeugt sein.

Zu beachten ist beim Tooleinsatz, dass es zwischen den einzelnen Werkzeugen erhebliche Qualitätsunterschiede gibt. Klassiker wie das Tool PC-Lint sind bereits seit Ende der 1970er-Jahre im Einsatz und bieten einen Einstieg ins Thema. Nachteile von „günstigen“ Werkzeugen sind oft die fehlende oder unkomfortable Benutzeroberfläche und eine unzureichende Dokumentation.
Analysewerkzeuge im oberen Preissegment bieten dem Benutzer einen weitaus größeren Komfort indem sie Fehlermeldungen nach Kritikalität sortieren. Das Entwicklerteam kann sich somit zunächst der Behebung der schwer wiegenden Fehler widmen und weniger kritische Meldungen dann bearbeiten, wenn hierzu noch Zeit bleibt.

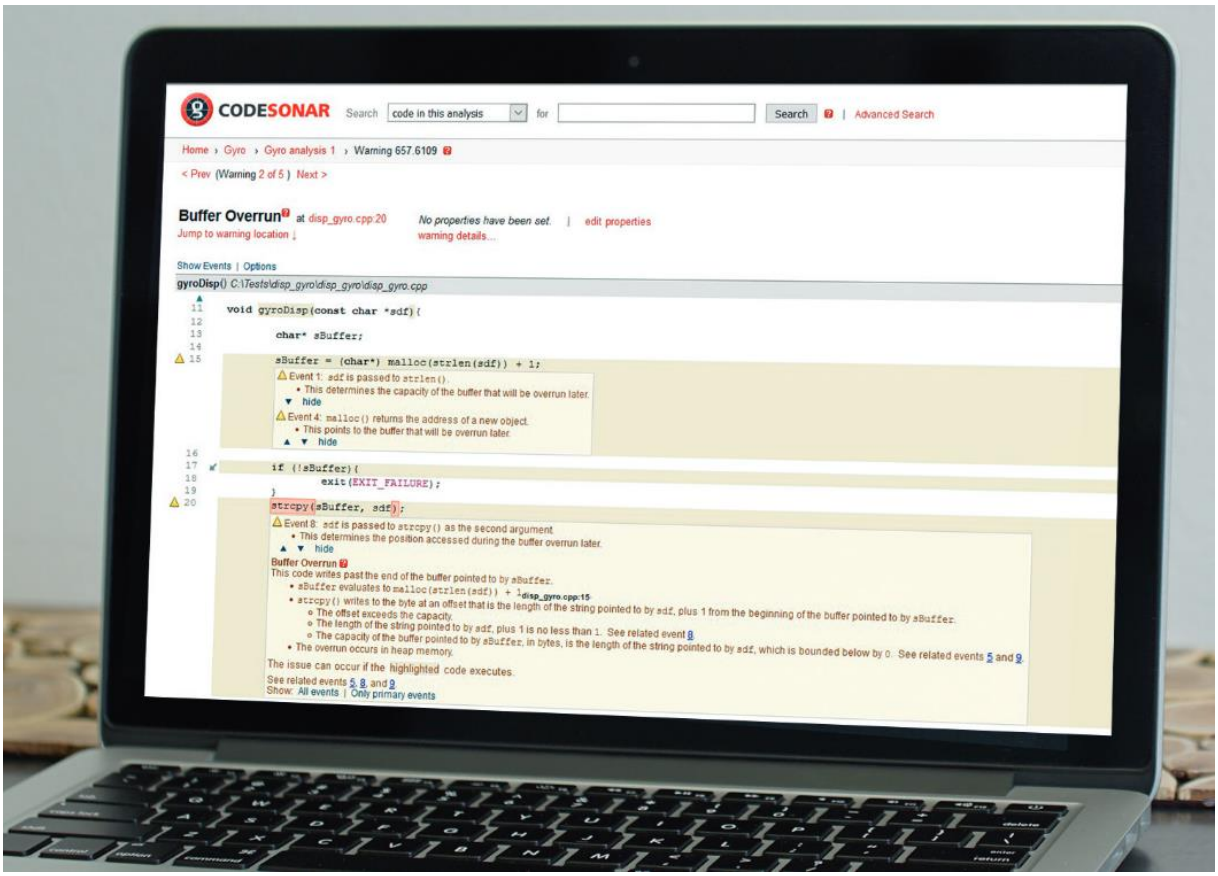

Abb. 3: Das Statische Code-Analysetool GrammaTech CodeSonar beschreibt Fehlermeldungen explizit und ermöglicht eine Klassifizierung der Meldungen.

Fehlermeldungen werden klassifiziert und explizit beschrieben. Zum Fixen können die Meldungen bestimmten Mitarbeitern zugewiesen werden. Warnungen, die vom Team als nicht relevant angesehen werden, können sogar ganz unterdrückt werden.

Der Einsatz von „Advanced Static Analysis Tools“ wie GrammaTech CodeSonar führt daher bei Entwicklern und Managern zu höherer Akzeptanz. Die etwas höheren Kosten für die Toolanschaffung werden in der Regel schnell wieder herausgeholt.

Wissen, was getestet ist: Code Coverage

Ist im fortgeschrittenen Projektstadium die Software dann ablauffähig, ist es sinnvoll, die statische Analyse durch dynamische Tests zu ergänzen. Bei sicherheitskritischer Software – also Software bei der im Problemfall Menschen zu Schaden kommen können – sind dynamische Testverfahren und der Nachweis der Testabdeckung (also der Code Coverage) aus gutem Grund verpflichtend. So fordern die DO-178C in der Luftfahrt, die ISO 26262 im Automobilbereich, die EN 50128 im Schienenverkehr und die allgemeine Norm IEC 61508 für Software mit hoher Kritikalität hohe Code-Coverage-Stufen wie die Modified Condition Decision Coverage (MC/DC), die den Test aller Bedingungen bzw. Entscheidungen in einer Software nachweist. Hierbei ist die Code Coverage so weit wie möglich auf dem Zielsystem zu prüfen, was nur mit leistungsfähigen Coverage-Analysern wie Testwell CTC++ möglich ist. Neben der Analyse aller geforderten Coverage Stufen bietet Testwell CTC++ den Vorteil mit allen Compilern und allen embedded Targets zu funktionieren.

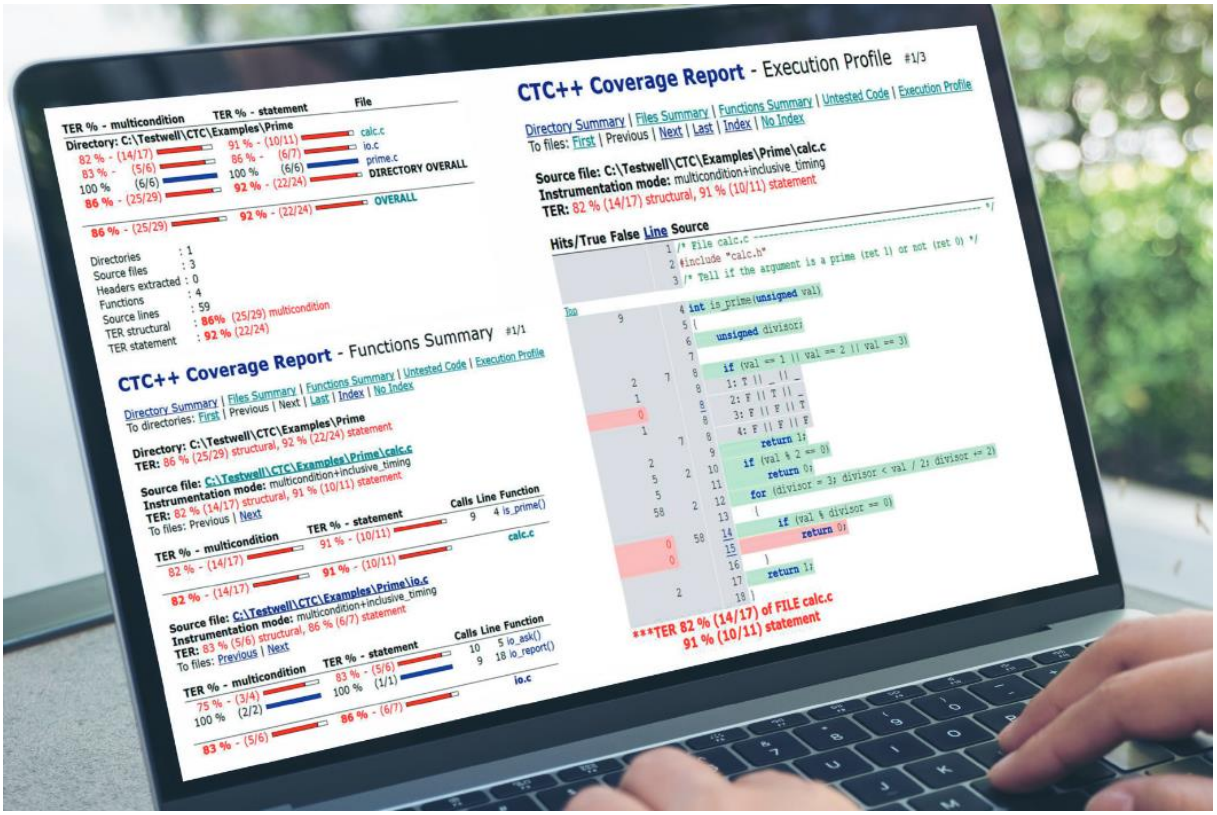

Abb. 4: Code Coverage Analyser wie Testwell CTC++ zeigen, welche Codeteile schon getestet sind und wo noch Tests nachgeliefert werden müssen.

Die Code Coverage zeigt genau, welche Tests noch fehlen, so dass der Tester gezielt die noch fehlenden Prüfungen initiieren kann. Auch können durch die Analyse der Testabdeckung redundante Tests, die zusätzliche Zeit kosten, vermieden werden.

Wiederverwendbare Software hilft Zeit und Kosten zu sparen

Eine Möglichkeit dem Marktdruck zu begegnen ist das Wiederverwenden von bereits existierender Software. Hierbei ist aber darauf zu achten, dass die in das neue Produkt eingearbeiteten alten Elemente verständlich und möglichst frei von Mängeln sind. Oft ist es aber leider so, dass über die Jahre viel Software ohne korrekte Qualitätskontrolle entwickelt wurde und sich versteckte Mängel angehäuft haben. Die Entwickler, die die Software vor Jahren geschrieben haben, sind oft nicht mehr greifbar. Erfahrungsgemäß kann man auch nicht auf gute Kommentierung oder Dokumentation zählen. Aber auch hier muss guter Rat nicht teuer sein: Analysetools wie Imagix 4D helfen, alte Software und Legacy-Code zu verstehen und eventuell notwendige Refactoring-Aktivitäten einzuleiten.

Imagix 4D ermöglicht eine einfache Beurteilung der Code-Qualität durch diverse statische Prüfungen und Erhebung von Metriken. Das automatisiert die Analyse des Kontrollflusses und der Abhängigkeiten. Das Werkzeug deckt Probleme in der Datennutzung und bei Task-Interaktionen auf.

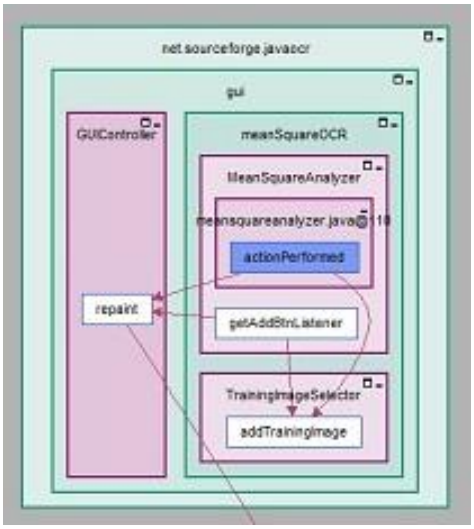

Abb.5: Werkzeuge wie Imagix 4D zeigen Abhängigkeiten von Codeteilen. Dies trägt zum besseren Verständnis des Codes bei und erleichtert Refactoring-Aktivitäten.

„Das Kind ist in den Brunnen gefallen“ – was tun?

In bereits länger laufenden Projekten ist es oft so, dass diese nicht entwicklungsbegleitend getestet wurden. Die Software ist geschrieben, qualitätssichernde Maßnahmen wurden aber weitgehend versäumt. Aus unserer Erfahrung kein Einzelfall – aber was tun?

Wurde die Integration eines statischen Analysetools in den Entwicklungsprozess versäumt, ist es trotzdem besser das fertige Programm statisch zu untersuchen, statt es „blind“ auszuliefern. Wegen der zu erwartenden Flut von Meldungen, macht es dann allerdings Sinn, sich auf die schwer wiegenden Fehler zu beschränken.

Selbst die Überprüfung von Software, die bereits in Betrieb ist, macht mitunter Sinn. So haben wir bei der Analyse einer Software, die bereits zehn Jahre in Medizingeräten im Einsatz war, noch zahlreiche kritische Fehler gefunden, die glücklicherweise bis dahin noch zu keinen Personenschäden geführt hatten. Selbst bei vermeintlich sauberer Software war die Ratio ein Fehler pro tausend Codezeilen eher die Regel als die Ausnahme. Wenn hierdurch Menschen zu Schaden kommen und der Softwarehersteller verklagt wird, dürfte das Gericht danach fragen, ob das Programm nach dem „Stand der Technik“ entwickelt und getestet wurde…

Hier gilt bezüglich der Softwaretests die Devise „besser spät als nie“. Auch wenn die Fehlerbehebung kurz vor bzw. nach Auslieferung deutlich teurer ist als während des Entwicklungsprozesses, gilt es, mit statischer Codeanalyse „zu retten was zu retten ist“.

Fazit

Softwaretest- und -analysen sind kein zusätzlicher Kostenfaktor, sondern ein Schlüssel zum Projekterfolg. Richtig eingesetzt, bieten Tools für die statische und dynamische Codeanalyse erhebliches Potential zur Kosteneinsparung und zur Steigerung der Produktivität.

Autoren

Klaus Lambertz ist Gründer und Geschäftsführer der in Offenburg ansässigen Verifysoft Technology GmbH. Verifysoft unterstützt über 600 Kunden mit Softwaretest-Werkzeugen in 40 Ländern weltweit.

Dipl.-Ing**. Royd Lüdtke** leitet bei Verifysoft Technology den Bereich Pre-Sales und Support für Statische Codeanalysetools. Er hat umfangreiche Erfahrung als Applikationsingenieur und Berater, hält mehrere Patente und ist Autor von Veröffentlichungen im IT-Bereich.

