Statische Analyse:

Sicherheitskritische Software bezahlbar

Software wird in immer stärkerem Maße für sicherheitskritische Aufgaben eingesetzt. Während die Entwicklungskosten pro Codezeile gleich bleiben, steigt der Umfang des Codes exponentiell an. Mit geeigneten Entwicklungswerkzeugen lässt sich das Dilemma aber lösen.

Von Dr. Paul Anderson

Wachsende Komplexität und die zunehmende Abhängigkeit von Software bei der Ausführung einsatzkritischer Funktionen hat dazu geführt, dass man bei der Entwicklung sicherheitskritischer Software an eine Grenze in puncto Bezahlbarkeit gestoßen ist [1]. Die Produktivität der Software-Entwickler bei sicherheitskritischen Systemen hat sich mit fünf Programmzeilen (Lines of Code; LOC) pro Tag bzw. rund 1000 LOC pro Jahr kaum verändert [2]. In die Höhe geschossen ist dagegen der Codeumfang sicherheitskritischer Software infolge der zunehmenden Abhängigkeit von eben dieser Software. Die Bezahlbarkeit von Software ist ein echtes Problem, und nicht nur verschiedene Standards, sondern auch Experten im Bereich der sicherheitskritischen Software empfehlen die Verwendung von Statische-Analyse-Tools, um diesem Problem beizukommen.

Die Software hat sich zum dominierenden Kostenfaktor sicherheitskritischer Systeme entwickelt. Tatsächlich entfällt heute ein Drittel der Kosten eines neuen Flugzeugs auf die Software und ihre Entwicklung. Bei den Automobilen gehen 25 % der Kapitalaufwendungen für neue Fahrzeuge auf das Konto der Elektronik und der Software. Zweifellos sind durch Software beeindruckende neue Möglichkeiten entstanden. Ihr

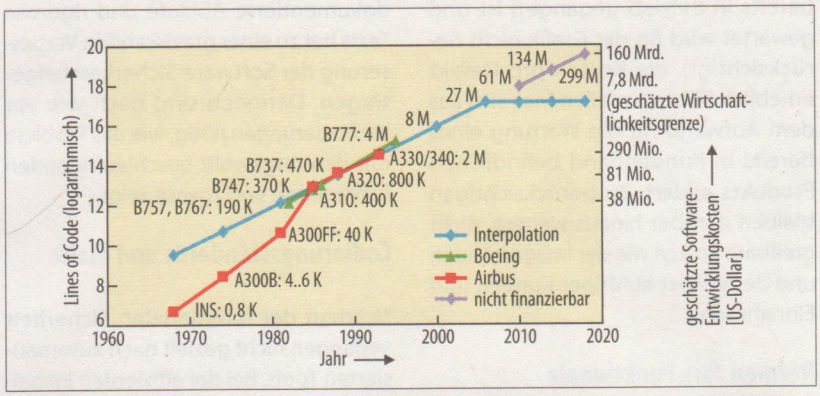


Bild 1. Untersuchungen verschiedener Flugzeug-Entwicklungsprojekte zeigen, dass sich der Umfang des Quellcode bei der Software etwa alle vier Jahre verdoppelt. (Quelle: [1])

exponentielles Wachstum und die damit zusammenhängenden Kosten aber haben dafür gesorgt, dass sie effektiv unbezahlbar geworden ist (**Bild 1**).

Klar ist, dass es mit den wachsenden Entwicklungskosten für größere Systeme nicht so weitergehen kann. Verschiedene Vorgehensweisen und Techniken wurden als Abhilfe vorgeschlagen. Dazu gehört die Verwendung von Tools – insbesondere von Statische-Analyse-Tools – mit dem Ziel, die Testabdeckung zu verbessern und Defekte aufzudecken, die von traditionellen Tests nicht erfasst werden. Das Software Engineering Institute SEI [4] und die NASA [5] empfehlen die statische Analyse als Hilfsmittel bei der Entwicklung sicherheitskritischer Software.

Exponentielle Kosten durch Fehler

Die Kosten, die entstehen, wenn Software-Fehler erst in bereits produzierten und ausgelieferten Produkten entdeckt werden, sind ebenfalls immens hoch. Rückrufaktionen, Haftungsprobleme und Imageschäden gehören zu den möglichen Konsequenzen. Nehmen wir Toyota als Beispiel: Die vom Drive-by-Wire-System des Unternehmens ausgelöste, ungewollte Beschleunigung verursachte Kosten von bis zu 5 Mrd. US-Dollar an Entschädigungen und Einnahmeverlusten [5] – zweifellos ein herber Verlust und eine wichtige Lektion in Sachen Software-Sicherheit. Auch wenn dieses Beispiel extrem erscheinen mag, wird doch deutlich, dass ein ernster Software-Fehler nahezu grenzenlose finanzielle Folgen haben kann. Studien haben ergeben, dass die Beseitigung von Software-Fehlern während der Produktion 100-mal mehr kostet als in den frühen Phasen der Entwicklung.

Wie von Capers Jones beschrieben [6], ist es irreführend, allein auf die Kosten pro Fehler zu blicken, denn hierbei bleibt die Menge der Defekte ebenso unberücksichtigt wie die Tatsache, dass die Kosten zum Finden und Beheben eines Defekts über die Zeit häufig gleichbleibt – worauf Entwickler gern verweisen. Bei den sicherheitskritischen Embedded-Systemen sind die Reparaturkosten allerdings höher als in anderen Branchen. Wird ein sicherheitskritischer Defekt nicht rechtzeitig behoben – oder schlimmstenfalls sogar absichtlich verborgen – so können die finanziellen Folgen durch gesetzliche Haftung zunehmen und Auswirkungen auf künftige Einnahmen haben.

Anhand des in **Bild 2** dargestellten V-Modells lässt sich der typische Software-Entwicklungs-Lebenszyklus und die relativen Vorteile der statischen Analyse in den einzelnen Entwicklungsphasen deutlich machen. Das V-Modell, das man in vielen Zertifizierungs-Standards für sicherheitskritische Software

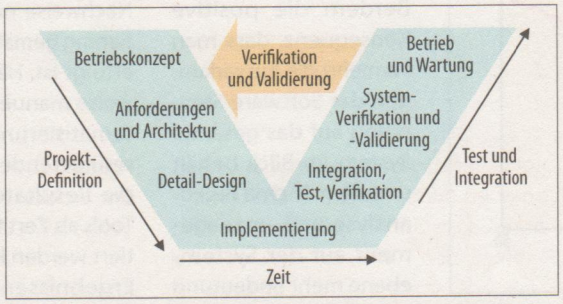


Bild 2. Auf dieses V-Modell für die Systementwicklung wird in Safety-Standards (z.B. ISO 61508 und 26262) häufig Bezug genommen.

(z.B. ISO 61508 und 26262) findet, ist hier ein gutes Beispiel.

Betrachtet man auf traditionelle Weise die Kosten pro Defekt über die Zeit oder in verschiedenen Phasen, so ist dies laut Jones zwar mathematisch korrekt, geht aber an der Praxis vorbei. Aussagefähiger sind stattdessen die Kosten pro Defekt bezogen auf das relative Volumen an Defekten. Letzteres verringert sich mit der Zeit und von einer Entwicklungsphase zur anderen, wie nachfolgend deutlich wird.

Das Interessante an **Bild 3** (beachten Sie die Reihenfolge der Entwicklungsphasen) ist, dass zwar die Kosten pro Defekt von einer Phase zur nächsten erwartungsgemäß zunehmen, dass die Gesamtkosten jedoch zurückgehen, weil die Zahl der Fehler insgesamt geringer wird. In der Praxis nimmt der Zeitaufwand zum Aufdecken und Beseitigen der Fehler nicht zu, aber die Kosten bleiben trotz der geringeren Fehlerhäufigkeit bestehen. Hervorzuheben ist ferner, dass bei einem Produkt, das bereits in Betrieb gegangen ist und gewartet wird (in der Grafik nicht berücksichtigt), die Kosten pro Defekt erheblich höher ausfallen, was sich aus dem Aufwand für die Wartung eines bereits in Kundenhand befindlichen Produkts erklärt. Zu berücksichtigen bleiben darüber hinaus weitere, nicht greifbare Kosten wie der Imageschaden und der Verlust künftiger Kunden und Einnahmen.

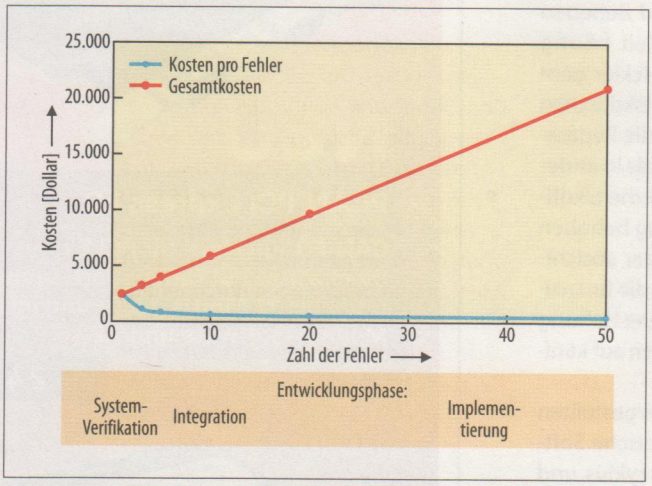


Bild 3. Gesamtkosten und Kosten pro Defekt bei abnehmender Defektanzahl.

(Quelle: [6])

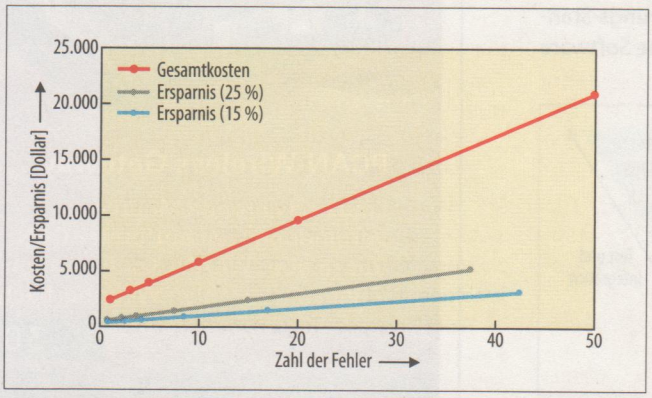


Bild 4. Einsparungen und Gesamtkosten durch die Reduzierung der Anzahl der in den einzelnen Phasen entstehenden Fehler.

Normen fürs Funktionale

Die funktionale Sicherheit muss ganzheitlich betrachtet werden. Die Sicherheit einer Komponente oder eines Subsystems wird also bezogen auf das gesamte System, einschließlich Hard- und Software, evaluiert. Dies ist ein durchaus wichtiger Aspekt vor dem Hintergrund der Tatsache, dass man die Software einst als unabhängigen oder gar unbedeutenden Bestandteil eines Systems ansah. Der zu trauriger Berühmtheit gelangte Fall Therac-25 [7] machte auf fatale Weise deutlich, dass diese Betrachtungsweise falsch war. Der Aerospace-Sektor wurde mit Normen wie z.B. DO-178 zum Vorreiter, was sowohl der Komplexität der verwendeten Software als auch der Sicherheitsrelevanz der entwickelten Systeme zuzuschreiben ist. Die Industrie, der Transportsektor, die Schienenverkehrstechnik und die Automobilbranche folgten im Lauf der Jahre mit Normen, die meist von der EN/ISO 61508 abgeleitet wurden.

Die verstärkte Fokussierung auf die funktionale Sicherheit hat außerdem die positive Konsequenz, dass man vermehrt die Auswirkungen der Software-Steuerung auf das gesamte System im Blick behält und dem Thema Risikoanalyse und -management auf der Systemebene mehr Bedeutung beimisst. Die Konzentration auf reproduzierbare, dokumentierte Abläufe und rigorose Tests hat zu einer gravierenden Verbesserung der Software-Sicherheit beigetragen. Dennoch sind nach wie vor Verbesserungen nötig, wie das Problem mit den ungewollt beschleunigenden Fahrzeugen von Toyota zeigt.

Codierungsstandards und mehr

Normen der funktionalen Sicherheit verlangen nicht gezielt nach automatisierten Tools. Bei der effizienten Einhaltung der Zertifizierungs-Anforderungen bieten Tools jedoch hervorragenden Nutzen. Zum Beispiel gibt die Norm ISO 26262 (Funktionale Sicherheit für Straßenfahrzeuge) Design- und Implementierungsprinzipien und Codierrichtlinien für Software-Module vor. Besonders nützlich sind Statische-Analyse-Tools bei der Durchsetzung von Codierungsstandards wie MISRA C.

So sinnvoll die Hilfestellung bei Codierungsstandards auch ist, stellt sie doch nur einen Bruchteil der Fähigkeiten eines Produkts wie CodeSonar von GrammaTech dar. Zertifizierungs-Standards verlangen nach Robustheit, Korrektheit und Folgerichtigkeit. Dies wiederum bedingt, dass beim Design, der Codierung und dem Testen eine über die Codierungs-Standards hinausgehende Stringenz angewendet werden muss. Statische-Analyse-Tools können Fehler im Quellcode vor und nach dessen Eingliederung in das Projekt entdecken. Ebenso können die Tools Fehler detektieren, die bei Tests schwierig zu finden und nur mit hohem Kostenaufwand zu beseitigen sind. Außerdem lässt sich auf manuellem Weg nur schwierig gewährleisten, dass Komplexität vermieden und die Pflegbarkeit verbessert wird. Tools wie GrammaTechs Code Browser CodeSurfer aber leisten äußerst wirksame Hilfestellung beim Management der Codestruktur.

Um Software zu zertifizieren, sind Nachweise nötig, dass die Implementierung gemäß dem jeweiligen Standard erfolgt ist. Häufig werden diese Nachweise manuell generiert, doch die Automatisierung verringert den daraus resultierenden Arbeitsaufwand. Damit die Resultate eines automatisierten Tools als Zertifizierungsnachweis akzeptiert werden können, muss man diesen Ergebnissen vertrauen. Zu diesem Zweck können die Tool-Anbieter auch ihre eigenen Produkte zertifizieren

lassen. Angesichts dieser Notwendigkeit wurde CodeSonar unabhängig gemäß ISO 26262, IEC 61508 und EN 50128 zertifiziert. Entwickler können die Tools somit im Vertrauen darauf verwenden, dass die produzierten Resultate für die Zertifizierungsstellen akzeptabel sind. Die Verwendung nicht qualifizierter Tools hingegen ist schlicht zu riskant, denn sie macht weitere Tests und Dokumentationen erforderlich und lässt die Zertifizierungskosten ansteigen.

Tools steigern Produktivität

Statische-Analyse-Tools ergeben greifbare Produktivitätssteigerungen für Software Teams, die eine Software-Sicherheits-Zertifizierung anstreben. Es hat entscheidende Vorteile, wenn man beginnend mit den ersten Entwicklungsphasen ein qualifiziertes Tool in den Software-Entwicklungsprozess einbezieht:

Die Codeabdeckung ist nicht alles: Viele Sicherheitsnormen verlangen nach einem hohen Grad an Codeabdeckung, also dem Nachweis, dass die meisten oder möglichst alle Anweisungen und Bedingungen geprüft wurden. Dies ist zwar sehr gründlich, aber auch sehr teuer und muss in jeder größeren Entwicklungsphase wiederholt werden (Modul-, Integrations- und Systemtest). Der Abdeckungsgrad wird davon bestimmt, wie kritisch die jeweilige Software ist. So kann weniger kritische Software (z.B. für das Bordunterhaltungssystem in einem Flugzeug) völlig ohne formelle Testabdeckung auskommen. Die Codeabdeckung ist eine Maßzahl, an der sich die Software-Qualität beurteilen lässt, doch es gibt Fälle, in denen hiermit nicht alles erfasst werden kann.

Fehler, die durch die Codeabdeckungs-Tests nicht erkannt werden: Nebenläufigkeitsfehler und Sicherheitslücken sind zwei wichtige Beispiele für Defekte, die auch von rigorosen Codeabdeckungs-Tests übersehen werden können. Die Nebenläufigkeit erweist sich häufig als schwierig zu programmieren und kann Fehler wie z.B. Race Conditions generieren, die so lange unentdeckt bleiben, bis während des Betriebs eine ganz bestimmte Bedingung vorliegt. Sicherheitslücken manifestieren sich als Fehler im Code. Die Bedingungen, die den Fehler hervorrufen, resultieren dagegen oft aus Eingaben, die während der Tests nicht berücksichtigt wurden.

Die statische Analyse kann derartige Fehler frühzeitig aufdecken und dadurch verhindern, dass sie in einer späten Entwicklungsphase zum Problem werden.

Frühzeitige Defekterkennung: Rigorose Tests können die meisten Defekte erkennen; sie sind aber teuer und äußerst zeitaufwändig. Das Erkennen und Vermeiden dieser Fehler gleich beim Schreiben des Codes ist bedeutend billiger als in einer späteren Entwicklungsphase. Die Kosten der Fehlerdetektierung nehmen mit der Zeit exponentiell zu. Die statische Analyse ist in der Lage, Bugs im Code als Bestandteil der Entwicklungsumgebung des Entwicklers sofort während des Programmierens zu entdecken, wodurch die fehlerbedingten Kosten in späteren Phasen drastisch gesenkt werden.

Analyse von zugelieferter Software: Dass bei der Entwicklung von Embedded Software auf kommerzielle und Open-Source Software zurückgegriffen wird, ist ein Fakt. Einige Sicherheitsstandards stufen jede Software, die nicht gemäß dem jeweiligen Standard entwickelt wurde, als SOUP (Software Of Unknown Pedigree), also als Software unbekannter Herkunft ein, die vor der Einbindung in das System genau untersucht werden muss. Statische-Analyse-Tools können zugelieferte Software und Binarys analysieren, um Defekte und Sicherheitslücken zu entdecken, die sich mit andersartigen Tests nicht würden detektieren lassen – sondern nur durch Einbinden und Ausführen, was eine teure Option ist.

Beschleunigte Zertifizierungsnachweise: Statische Analysen – ebenso wie viele weitere Test- und Lifecycle Management Tools – sorgen für eine automatisierte Dokumentation zur Unterstützung von Tests, Codierungsstandards und Qualitäts- bzw. Robustheitsnachweisen. Ein Großteil der Arbeitszeit bei der Sicherheits-Zertifizierung geht auf das Konto der Dokumentation und des Erbringens von Nachweisen. Die Automatisierung und speziell die statische Analyse reduzieren diesen Aufwand deutlich.

Statische Analyse zahlt sich aus

Wie sieht es angesichts dieser Faktoren mit der Rentabilität aus? Die statische Analyse verringert die Zahl der Defekte in der entwickelten Software in sämtlichen Entwicklungsphasen. Eine einfache Analyse besteht darin, die Zahl der Defekte gemäß den Daten aus **Bild 4** zu reduzieren. Aus dieser geringeren Zahl entstandener Fehler während der Entwicklung lässt sich eine gravierende Kostensenkung ableiten.

Diese einfache Auswertung ergibt Einsparungen von rund 126 US-Dollar pro Defekt, wenn man von durchschnittlich 15 Defekten pro 1000 LOC ausgeht (während der Entwicklung bei hohem Defektvolumen). Hieraus errechnen sich Einsparungen von 1900 US-Dollar pro 1000 LOC. Selbstverständlich variieren die Ergebnisse infolge anderer Faktoren wie Lohnkosten, Zeitaufwand für Defekterkennung und -beseitigung und Defektdichte. Berücksichtigt man aber, dass viele sicherheitskritische Systeme 100.000 LOC und mehr enthalten, ist die Wirtschaftlichkeitsberechnung für die statische Analyse klar. Dabei sind hier noch nicht einmal die Kosten nach erfolgtem Einsatz beim Kunden berücksichtigt, die wie erwähnt erheblich höher sind.

Neben der Defekterkennung wird CodeSonar zum Aufdecken komplexer Nebenläufigkeitsprobleme, zum Analysieren von Third-Party Software und -Binarys sowie zur Erkennung von Fehlern benutzt, die von traditionellen Tests übersehen werden. Diese kritischen Aspekte sind bei der oben durchgeführten, eher einfachen Analyse nicht berücksichtigt, tragen aber eindeutig zur Rentabilität des Tools bei. Allerdings beschert das Finden sonst unentdeckt gebliebener Defekte dem Entwicklungsteam die größten Vorteile.

Kosten im Griff behalten durch weniger Fehler

Die Entwicklung sicherheitskritischer Software wird immer teurer, und so suchen die Hersteller nach Lösungen zur Verbesserung der Entwicklerproduktivität. Statische-Analyse-Tools stellen hier unerlässliche Hilfsmittel dar und werden von Experten auf diesem Gebiet zu Stützpfeilern ihrer Software-Entwicklung gemacht.

Trotz rigoroser Tests kommen in sicherheitskritischer Software immer noch Fehler vor – mit teils katastrophalen menschlichen und wirtschaftlichen Folgen. Im Bereich der sicherheitskriti-

schen Software sind Statische-Analyse-Tools unerlässlich, um die Entwicklung sicherer und qualitativ hochwertiger Software zu gewährleisten. In einigen Fällen empfehlen Sicherheitsnormen die Verwendung von Statische-Analyse-Tools zum Aufdecken von Fehlern, die bei Tests möglicherweise übersehen werden, sowie – neben anderen Vorteilen – zur Durchsetzung von Codierungsstandards. Die Rentabilität von Statische-Analyse-Tools unterstreicht die Tatsache, dass die statische Analyse eine wichtige Rolle nicht nur bei der Entwicklung, sondern auch bei der Markteinführung spielt.

Durch die Steigerung der Entwicklungs- und Testproduktivität und das Auffinden von Defekten, die von herkömmlichen Tests nicht erkannt werden, spielt die statische Analyse eine Schlüsselrolle bei der Lösung des Bezahlbarkeits-Problems sicherheitskritischer Software. *jk*

Literatur

[1] *Redman, D. et al.*: Virtual Integration for Improved System Design, Carnegie Mellon University, Pittsburgh, USA, 2010. wiki.sei.cmu.edu/aadl/images/d/de/SAVI_Virtual_Integration-AVICPS2010.pdf

[2] *Benediktsson, O.*: Safety critical software and development productivity. Proceedings for The Second World Congress on Software Quality, Yokohama, O., 2000. www.eis.mdx.ac.uk/research/SFC/Papers/2WCSQPreprint.pdf

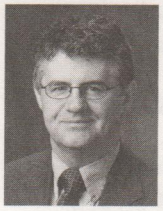
[3] *Feiler, P., et al.*: Four Pillars for Improving the Quality of Safety-Critical Software Reliant Systems. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, 2013. resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_47803.pdf

[4] *Holzmann; G.*: The Power of Ten – Rules for Developing Safety Critical Code. NASA/JPL Laboratory for Reliable Software, Pasadena, USA. spinroot.com/gerard/pdf/P10.pdf

[5] Foster School of Business: Accounting for Toyota's Recalls. 2011. faculty.washington.edu/rbowen/cases/Toyota_Recall_case_April_2011.pdf

[6] *Jones, C.*: A Short History of the Cost per Defect Metric. 2012. www.ifpug.org/Documents/Jones-CostPerDefectMetric-Version4.pdf

[7] Wikipedia: The Therac-25 Incident. en.wikipedia.org/wiki/Therac-25



Dr. Paul Anderson ist Vice President Engineering bei GrammaTech (USA), einem Hersteller von Statische-Code-Analyse-Tools. Dr. Anderson hat über 20 Jahre Erfahrung in der Entwicklung von Statische-Analyse-Werkzeugen und automatisierten Test Tools. Er besitzt einen Hochschulabschluss (Bachelor of Science) des Kings College, Universität London, und einen Doktortitel der City-Universität London. Seine Arbeit findet Erwähnung in zahlreichen Artikeln, Buchkapiteln und auf internationalen Konferenzen.

sales@grammatech.com

Post-Mortem-Analyse:

Letzte Lebenszeichen

Fehler, die nur sporadisch auftreten, sind besonders schwer zu finden. Mit seinem SystemView zeigt Segger eine Methode zur Ursachenforschung auf, die sich sogar in Seriengeräten anwenden lässt.

Selbst bei Serienprodukten, die bereits im Feld im Einsatz sind, kommt es vor, dass sie lange Zeit zuverlässig arbeiten und dann ohne ersichtlichen Grund ausfallen. In den meisten Fällen lässt sich die Ursache nicht mehr ermitteln, wenn das System nicht an einen Debugger angeschlossen war. Mit dem Werkzeug SystemView von Segger kann man dem Problem aber auf die Spur kommen. Die Echtzeit-Aufnahmen von SystemView können nicht nur zum Debuggen und Testen eingesetzt werden. Durch den sehr geringen Overhead können die Aufnahmen auch im Final Release durchgeführt werden.

Im Post-Mortem-Modus werden System-Events kontinuierlich im RAM-Puffer des Systems aufgenommen. Hierbei wird ein Ringpuffer verwendet, sodass die letzten Events erhalten bleiben. Mit dem J-Link oder jeder anderen Debug Probe, die sich an ein laufendes System anschließen lässt, kann der SystemView Buffer ausgelesen werden. Das heißt, dass für die Aufnahme keine Debug-Verbindung aktiv sein muss. Die Debug Probe kann bei Bedarf jederzeit angeschlossen werden.

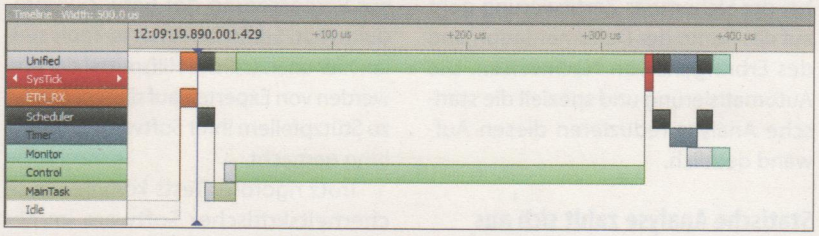


Bild 1. Kurz vor dem Absturz war die Steuerungs-Task (grün) aktiv, um das Heizsystem herunterzufahren. Dann trat ein Timer Interrupt auf, um die Monitor-Task zu starten. Durch einen Ressourcenkonflikt bringt diese Konstellation das System zum Absturz.

Crash-Analyse

Im Falle eines Systemabsturzes liefert SystemView wichtige Informationen über das Systemverhalten kurz vor dem Absturz. Die enthaltenen Informationen tragen dazu bei, das Problem zu lösen, oder helfen bei der Erstellung eines Reproduktionsszenarios.

Die Arbeitsweise lässt sich an einem Beispiel gut verdeutlichen: Ein Smart-Home-System steuert die Zentralheizung und ist mit dem Internet verbunden. Es erhält Befehle von einem Steuerungs-Server. Die Befehle werden durch eine Steuerungs-Task verarbeitet. Im Wesentlichen arbeitet das System zuverlässig, aber einige Geräte stürzen in unregelmäßigen Abständen ab. Beim Debuggen lassen sich diese Abstürze nicht nachvollziehen. Es gibt keinen Hinweis auf die Ursache.

Nun lassen sich mit dem Post-Mortem-Modus von SystemView Informationen sammeln, was direkt vor dem Absturz geschehen ist, also welche Task aktiv war, bevor das System abgestürzt ist.