



TECHNISCHE FACHHOCHSCHULE BERLIN
University of Applied Sciences



BOSCH

**Analyse und Umsetzung von Verbesserungspotentialen
bei Software-Integrationstests für Entwicklungsprojekte
in der Body Electronics Domäne
am Beispiel der Robert Bosch GmbH**

Master-Thesis

Im Fernstudiengang „Industrial Engineering“
der Technischen Fachhochschule Berlin
zur Erlangung des akademischen Grades eines Masters of Science

vorgelegt von

Heiko Frey

Oktober 2006

Abstract

Software integration testing can be divided into three sections: The static analysis, the symbolic execution and the dynamic test. While the static analysis exposes the couplings between the modules and its anomalies, the symbolic execution uncovers runtime errors, which only might happen under certain conditions. The dynamic test checks the interface operation, as it is specified in the software architecture and in the module specifications.

The software integration test is applied to a sample project of the body electronics domain. Here, oftenly no software architecture and no module specifications are used due to a small development team and the reuse of code from earlier projects. The interfaces between modules are commonly global variables.

While the static analysis, couplings between modules are identified. Anomalies within these coupling can be found, like unreachable functions, dead variables or reading of undefined variables. With the symbolic execution, unreachable code can be identified, as well as operations, which might fail according to external input values. A dynamic test is hardly possible, due to the missing architecture and module specifications. A work-around method is introduced, which concentrates on the interaction of two functions via a global single variable. The meaning of a variable is taken out of the code, the module design or the module test specification of the writing function. A test then determines if the reading function is acting as the writing function aimed by its writing.

Finally, recommendations are formulated to improve the integration testing process.

Zusammenfassung und Ausblick

Der Software-Integrationstest ist ein Abschnitt der Softwareentwicklung, der sich mit der Verifikation der Schnittstellen zwischen Softwaremodulen befasst. Er folgt auf den Modultest, bei dem die Funktionen und Module einzeln isoliert geprüft werden. Bei der Integration werden die Module gemäß einer Integrationsstrategie zu einem Softwaresystem zusammengefügt. Nach jedem Integrationsschritt folgt ein Integrationstest, bei dem das neu hinzugefügte Modul (oder mehrere neu hinzugefügte Module) auf das wunschgemäße Zusammenarbeiten mit dem Rest des (teilintegrierten) Softwaresystems hin untersucht wird.

Der Software-Integrationstest gliedert sich in drei Teile, eine statische Integrationsanalyse, eine symbolische Ausführung und einen dynamischen Integrationstest.

Der Software-Integrationstest wurde an einem Beispielprojekt der Karosserieelektronik durchgeführt. Die Softwareentwicklung in diesem Bereich ist geprägt durch kleine Entwicklungsteams mit kurzen Kommunikationswegen. Deshalb wird häufig auf die Erstellung einer Modulspezifikation und einer Softwarearchitektur verzichtet. Weiterhin kommunizieren die Funktionen und Module zumeist über globale Variablen. Basis der Entwicklung ist die Systemspezifikation. Für den Modultest wird daher vom Testteam ein Reengineeringprozess durchlaufen, bei dem ein Verständnis für die Funktionalität der Funktionen und Module geschaffen wird und Anforderungen der Systemspezifikation auf einzelne Funktionsgruppen heruntergebrochen werden können. Aus diesem Grund kann das Testteam dann auch einen intramodularen Test durchführen, der jedoch auf offensichtlich zusammengehörige Funktionen als einen Block (als ein Modul) auf ihre Funktionalität hin testet.

Um eine statische Software-Integrationsanalyse durchführen zu können, wurden die Kopplungen der Softwarefunktionen untereinander ermittelt. Hierzu wurden mit Hilfe des Tools QA-C auf Dateiebene die Funktionen und globalen Variablen ermittelt. Im Anschluss wurden mit einem selbst geschriebenen Programm die Bezüge der Funktionen und Variablen über Dateigrenzen hinweg ermittelt. Auf diese Weise konnten nicht korrekt genutzte Variablen und Funktionen ermittelt werden, sowie Variablen, die von unterschiedlichen Funktionen für verschiedene Zwecke eingesetzt werden.

Durch eine anschließende symbolische Ausführung konnten Codebereiche aufgedeckt werden, die niemals durchlaufen werden und Bereiche, die aufgrund äußerer Eingabewerte einen Laufzeitfehler erzeugen können. Als Tool wurde hierbei Polyspace eingesetzt, das das komplett integrierte Softwaresystem symbolisch analysierte. Die

externen Schnittstellen des Softwaresystems, wie z.B. Interruptquellen und externe Parameter wurden dabei simuliert. Die gefunden Laufzeitfehler können weder durch eine statische Analyse, noch durch einen dynamischen Test gefunden werden, da sie nur unter bestimmten Bedingungen auftreten.

Der dynamische Integrationstest kann nur stark eingeschränkt durchgeführt werden, weil keine Spezifikation der Softwarearchitektur, Module und Funktionen vorhanden ist. Die Möglichkeiten beschränken sich auf den funktionalen Integrationstest, ein Test des modulübergreifenden Kontroll- und Datenflusses, sowie der wertbezogene Test kann nicht durchgeführt werden. Es wurde ein Verfahren erarbeitet, dass durch Analyse des Codes, des Moduldesigns oder der Modultestspezifikation die Bedeutung einzelner globaler Variablen ermittelt und darauf getestet, ob das gewünschte Verhalten, dass eine schreibende Funktion mit der Wertzuweisung auf eine Variable äußert, durch die lesende Funktion auch eingehalten wird. Zur Erstellung der Testfälle können die bereits ausgeführten Modultestfälle herangezogen und kombiniert werden. Als unterstützendes Tool eignet sich wie auch für den Modultest Rational Test Realtime. Das Verfahren wurde an mehreren Beispielen durchgeführt. Dabei wurden keine funktionalen Fehler gefunden, aber einige Schwächen, wie z.B. parallel genutzte globale Variablen oder identische Funktionalitäten mehrerer Funktionen.

Zur Priorisierung der dynamischen Tests wurden zwei Metriken untersucht, die die Komplexität der mittels globaler Variablen realisierten Schnittstellen messen. Während eine Metrik die Schnittstellenkomplexität einer Funktion misst (*Global variables read*), misst die andere Metrik die Verwendungshäufigkeit einer globalen Variablen (*Functions reading + Functions writing*). Durch die beispielhaft durchgeführten Tests zeigte sich, dass Funktionen, die viele globale Variablen verwenden eher fehleranfällig sind als Funktionen, die wenige globale Variablen verwenden. Variablen, die nur von wenigen Funktionen verwendet werden, sind mitunter überflüssig.

Abschließend wurden Empfehlungen für eine Verbesserung der Prozessbeschreibung gegeben, um die Entwicklung und das Testteam bei der Durchführung von Software-Integrationstest besser zu unterstützen.

Um eine wirkliche Verbesserung herbeizuführen, die neben dem Integrationstest den gesamten Entwicklungsablauf verbessert, ist eine spezifizierte, den Projekten angepasste, schlanke Softwarearchitektur mit wenig Kopplung zwischen den Modulen unabdingbar. Generell muss daher für die Softwareentwurfsphase mehr Aufwand eingeplant werden. Gleiches gilt für eine detailliertere Dokumentation der Arbeitsergebnisse. Mit einer Erstellung der Dokumentation, insbesondere der entstandenen Architektur und der Moduldesigns, durch die Entwickler selbst, kann die

erarbeitete Softwarestruktur bereits kritisch hinterfragt werden, ohne dass ein Test durchgeführt wird.

Bislang werden in erster Linie aus Zeitgründen die Entwurfsphasen abgekürzt und direkt mit der Implementierung begonnen, um den Kunden so schnell wie möglich ein Muster zur Verfügung stellen zu können. Somit liegt ein Grund für die aktuelle Vorgehensweise auch bei den Kunden, die immer kürzere Entwicklungszeiten fordern, bei steigender Funktionalität und sinkenden Kostenvorstellungen. Da von dieser Seite aber kaum eine Entspannung der Situation erwartet werden kann, muss die Strategie bei der Produktentwicklung angepasst werden. Durch eine kundenunabhängige Plattformentwicklung können die Rahmenbedingungen (Architektur, Grundfunktionalität) geschaffen werden, um viele Kunden mit einem System beliefern zu können, dass die Anforderungen an Testbarkeit und Wartbarkeit erfüllt und gleichzeitig zügig an unterschiedliche Kundenanforderungen angepasst werden kann. Ansätze hierzu gibt es bereits in der Praxis wie die Herstellerinitiative Software, AUTOSAR und die kürzlich entwickelte Referenzarchitektur für EB-Softwareprodukte zeigen.