Automatischer Einklang von Architektur und Implementierung

Hand aufs Herz: Wer hat Lust auf manuelle Reviews?

Ralph Dittmar, TR-Electronic GmbH
Thomas Eisenbarth, Axivion GmbH

Die Architektur als globale Struktur unserer Software ist das Fundament unserer täglichen Arbeit. Daher ist es essentiell, dass wir uns mit unserer Architektur auseinandersetzen und nicht gegen sie arbeiten: Planen, Prüfen und Bewerten können wir am besten mit Blick auf die Architektur. Entwickler arbeiten klassisch jedoch sehr detailliert an ihrer Software und verlieren so leicht den Blick für das Globale. Daher haben wir bei TR-Electronic eine Architekturprüfung etabliert, die uns den Aufwand für diesbezügliche manuelle und ungeliebte Reviews erspart und die Architektur im Tagesgeschäft lebendig werden lässt. Von diesem Weg und den erzielten Resultaten möchten wir berichten, denn eingehaltene Architekturen sind für uns inzwischen eine Selbstverständlichkeit in unserem Entwicklungsprozess geworden.

Einführung

Die Architekturbeschreibung eines Software-Systems erlaubt, auf abstraktem Niveau über die Software zu sprechen und Erweiterungen an der Software zu analysieren, zu planen und zu bewerten. Damit dies gut funktioniert, muss die Architektur unter anderem zur Implementierung passen.

Eine Architektur im hier verwendeten Sinn beschreibt die statische Struktur des Systems: In welche Komponenten ist das System hierarchisch aufgeteilt. Die Beziehungen zwischen den Komponenten beschreiben die Regeln für die Verwendung von Schnittstellen einer Komponente durch die anderen Komponenten.

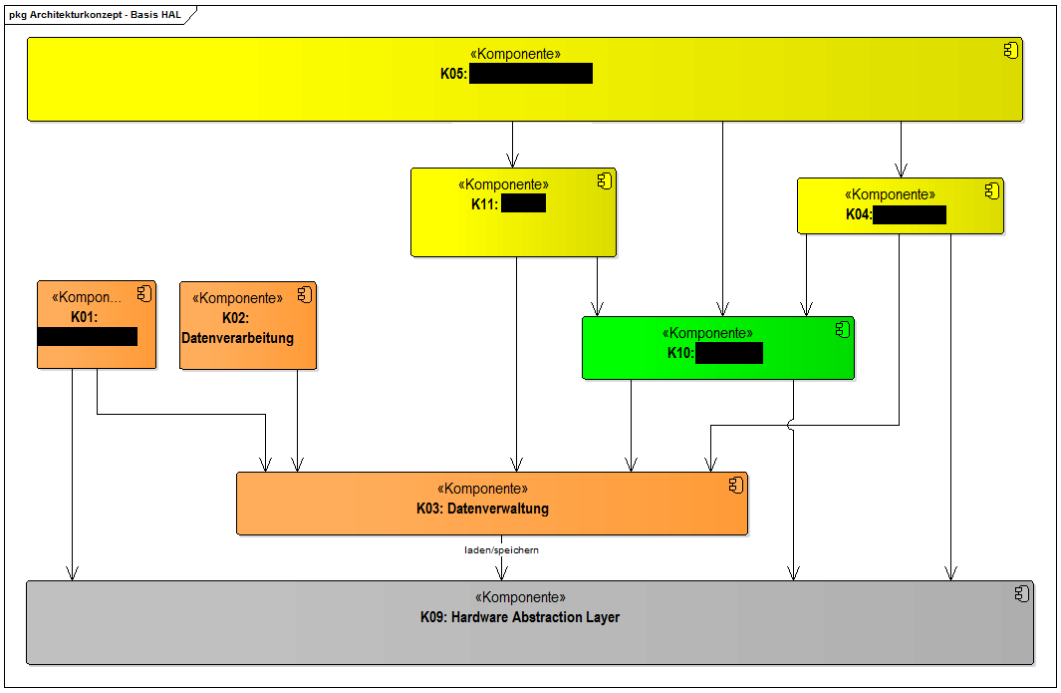

Abb. 1: Beziehungen der Komponenten zum HAL.

Im Beispiel in Abbildung 1 darf Komponente K11 auf Schnittstellen der Komponenten K03 und K10 zugreifen. K05 darf aber nicht auf K03 direkt zugreifen.

Auswirkungen einer nicht zur Implementierung passenden Architekturbeschreibung

Eine nicht zur Implementierung passende Architekturbeschreibung führt zu folgenden Effekten:

- Es kommt zu einem schleichenden Erosionsprozess. Dieser wird mit der Zeit immer stärker.
- Das Verständnis der Software leidet zunehmend.
- Der Respekt vor der Architektur schwindet („warum soll ich der sein, der sich an die Architektur hält…“).
- Die Planung wird sehr feinteilig, da die grobe Struktur nicht mehr passt und als Planungshilfsmittel nicht mehr taugt.

Es zahlt sich also aus, die Architektur und die Implementierung zueinander passend zu halten. Übrigens heißt dies nicht, dass eine einmal erzeugte Architektur sklavisch eingehalten wird. Es kann durchaus vorkommen, dass die Idee hinter der Architektur sich ändert und die Implementierung zu der geänderten Idee passt, aber vergessen wird, die Architekturbeschreibung entsprechend anzupassen. Dann ist die Lösung, die Architekturbeschreibung zu korrigieren. Man muss allerdings erkennen können, dass Implementierung und Architekturbeschreibung divergieren.

Nachteile manueller Architekturprüfung

Die Software hat einen Umfang von über 100 Modulen erreicht und wächst weiter, wir stehen also vor einem Mengenproblem. Ein manueller Prozess zur Prüfung ist immer fehlerbehaftet und zeitintensiv. Jede Änderung an der Implementierung oder an der Architekturbeschreibung erfordert eine wiederholte Prüfung. Der manuell notwendige Aufwand ist schlicht nicht zu leisten.

Durch die eingesetzte automatische Architekturprüfung, die entwicklungsbegleitend als Teil des Builds abläuft, bekommen wir direktes Feedback. Die Mitglieder des Projektteams werden im Sinne der Wartbarkeit „erzogen“. Verletzungen der Architektur werden immer aufgezeigt. Insgesamt ergibt sich mehr Sicherheit für den Entwickler: Habe ich meine Aufgabe strukturell richtig erledigt? Außerdem steigt die Transparenz für den Projektleiter: Liegt mein Projekt strukturell im Ziel oder sind Nacharbeiten notwendig? Die Möglichkeit menschlicher Fehler wird generell verringert.

Außerdem setzen wir dieselben Komponenten und Module in unterschiedlichen Anwendungen ein, so dass manuell mehr als nur eine Anwendung geprüft werden müsste. Durch die automatische entwicklungsbegleitende Prüfung wird nach einer Implementierungsänderung für alle Anwendungen sichergestellt, dass die Implementierung weiterhin zur Architekturbeschreibung passt.

Funktionsweise der automatischen Architekturprüfung

Die von uns eingesetzte automatische Architekturprüfung mit der Axivion Bauhaus Suite basierend auf hierarchischen Reflexionsmodellen [1] hat folgende Voraussetzungen:

- Es gibt eine Architekturbeschreibung (Graph), in unserem Falle in Enterprise Architect per UML-Model gepflegt.
- Es gibt eine Vorschrift für die automatische Erkennung des Zusammenhangs zwischen Implementierung und Architekturkomponenten. Im vorliegenden Fall ist dies per einfacher Namensregel gelöst. Dadurch ist die Übertragung der Architekturinformation auf die Implementierung auch für die Entwickler sehr einfach und nicht fehleranfällig.

Aus der Implementierung werden durch eine statische Analyse automatisch die Abhängigkeiten zwischen den Funktionen, Variablen, Typen, usw. abgeleitet. Für die Prüfung einer dieser Abhängigkeiten in der Implementierung gegen die Architektur gibt es drei mögliche Resultate: Konvergenz, Absenz und Divergenz. Im Falle einer Konvergenz stimmt die gefundene Abhängigkeit im Code mit der Forderung der Architektur überein. Eine Absenz liegt vor, wenn in der Architektur eine Abhängigkeit gefordert wird, diese in der Implementierung aber gar nicht vorliegt. Eine Divergenz schließlich liegt vor, wenn in der Implementierung eine Abhängigkeit gefunden wird, die durch die Architektur nicht erlaubt wird.

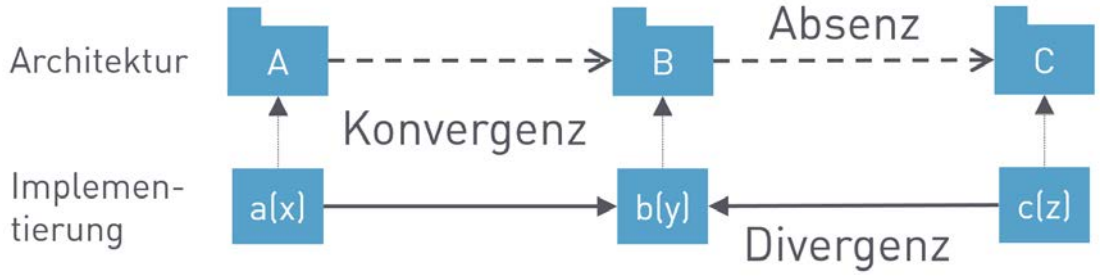

Abb. 2: Beispiel für eine Architekturprüfung. Beziehungen zwischen Funktionen sind Funktionsaufrufe im Code.

In Abbildung 2 sind drei Komponenten A, B und C dargestellt, denen jeweils eine Funktion a, b und c als Implementierung zugeordnet ist. Da A eine Beziehung zu B haben darf und a(x) die Funktion b(y) aufruft, ergibt sich eine Konvergenz. Die Beziehung von B nach C hat keine Entsprechung in der Implementierung, so dass sich eine Absenz ergibt. Der Aufruf von c(z) nach b(y) schließlich ist eine Divergenz, da ihn die Architektur nicht erlaubt (es müsste dafür eine Beziehung von C nach B geben).

Als Verstoß sind Divergenzen und Absenzen zu werten. Ein Verstoß kann durch ein Missverständnis und daraus resultierend einer Abweichung in der Implementierung begründet sein. Außerdem kann ein Verstoß aus einer in der Architekturbeschreibung nicht nachgezogen Architekturänderung resultieren, beispielsweise wenn es zu Planänderung während der Entwicklung kommt: Die Implementierung passt zur

Architektur, diese ist aber nicht mehr korrekt in der Architekturbeschreibung dargestellt.

Darstellung und Nutzen der Ergebnisse
Die gefundenen Verstöße werden dem Entwickler im Detail in einem Web-Dashboard der Axivion Bauhaus Suite angezeigt. Für den Projektleiter/Manager sind Übersichten vorhanden, aus denen der Verlauf der Verstöße und ihre absolute Anzahl während der Entwicklung ersichtlich ist. Im Dashboard werden außerdem weitere Erosionsindikatoren dargestellt, z.B. Stilverstöße, zyklische Abhängigkeiten und Toter Code, so dass ein Überblick über alle Verstöße in der zu entwickelnden Software verfügbar ist.

Ein Entwickler kann, je nach Grund für die gefundene Abweichung, sofort auf den Verstoß reagieren, z.B. indem er die Architektur nachzieht (oder einen entsprechenden Hinweis an den Architekturen gibt) oder die Implementierung so umgestaltet, wie es laut Architektur geplant war. Dadurch kommt es zu einer verstärkten Kommunikation über die Architektur und zu Lerneffekten. Des Weiteren erleichtern uns die aus der Architektur geforderten und nun kontrollierten Schnittstellen die Verteilung von Arbeitspaketen auf mehrere Entwickler.

Projektleiter und Manager haben die Möglichkeit, die Gesamtmenge an technischen Schulden in der Software zu bewerten und betriebswirtschaftliche Argumente zusammen mit den technischen Argumenten, also Schulden der Vergangenheit und geschätzte Aufwände der zukünftigen Entwicklung, im Blick zu behalten.

Folgen der Architekturprüfung für Prozesse und Ergebnisse
Wenn man code-zentriert arbeitet, ergeben sich durch die neue Betonung der Architektur und damit der globalen Zusammenhänge und der Struktur der zu entwickelnden Software ganz neue Perspektiven. Langfristige Entwicklungen werden auf diese Weise vereinfacht, da nicht jeder Arbeitsschritt eine Lektüre des Codes erzwingt. Da Entwickler sich im Vorfeld mit der Architektur beschäftigen, entstehen weniger Verstöße beim Entwickeln, unmögliche Forderungen der Architektur werden früher aufgedeckt und diskutiert.

Fazit
Durch die Architekturprüfung sind wir zu einer eher architekturbetonten Arbeitsweise übergegangen. Es entstehen gleich beim Entwickeln weniger Abweichungen zwischen Architekturbeschreibung und Implementierung. Die Entwickler werden entlastet, da es in der Implementierungsphase weniger Überraschungen gibt. Die automatisierte und entwicklungsbegleitende Architekturprüfung spart manuelle Arbeit, die wir ansonsten nur stichprobenhaft leisten könnten.

Literatur
[1] Koschke, Rainer; Simon, Daniel (2003): Hierarchical Reflexion Models. In: WCRE 03: Proceedings of the 10th Working Conference on Reverse Engineering. S. 36-45

Autoren

Ralph Dittmar ist Softwareentwickler bei der TR-Electronic GmbH und arbeitet seit 2013 an Software-Projekten für Produkte mit Zertifizierung bis SIL3/Ple. Herr Dittmar ist seit 15 Jahren Software-Entwickler im embedded-Bereich.
Email: ralph.dittmar@tr-electronic.de

Thomas Eisenbarth ist Gründer und Geschäftsführer der Axivion GmbH, die Analysewerkzeuge rund um das Thema Erosionsschutz von Software entwickelt und vertreibt. Herr Eisenbarth beschäftigt sich seit über 20 Jahren mit der Analyse von Software und der Prüfung von Software-Architekturen.
Email: eisenbarth@axivion.com