Das Paradoxon „kundenspezifische Standardsoftware“

Zwei Welten, zwei Lösungen und was wir voneinander lernen können

Matthias Essig, WITTENSTEIN electronics GmbH
Benjamin Boost, AIT GmbH & Co. KG

Individualisierung von Produkten ist ein zentrales Thema, welches bei nahezu jeder Art von Produkten und in jeder Branche zu finden ist. Somit ist es auch eine der Kernfragen in der Software-Entwicklung. Jeder Kunde möchte eine Software, die seine spezifischen Anforderungen erfüllt. Eine Standardsoftware, egal ob im Embedded- oder im Applikationsbereich, kann dies nicht leisten. Dennoch ist der Anteil an Komponenten, die in mehreren Kundenprojekten genutzt werden, in den meisten Fällen sehr hoch. Wie ist es nun also möglich, eine Wiederverwendung und Standardisierung mit der Fertigung von individuellen Produkten in Einklang zu bringen?

Sowohl die WITTENSTEIN electronics GmbH als auch die AIT GmbH & Co. KG mussten sich dieser Frage stellen. Dabei könnten Randbedingungen unterschiedlicher kaum sein. Die WITTENSTEIN electronics GmbH realisiert innovative, individuelle mechatronische Antriebslösungen, die ihren Kunden zu Alleinstellungsmerkmalen in ihren Märkten verhelfen. Die AIT GmbH & Co. KG hingegen ist ein Lösungspartner für Softwareentwicklung basierend auf Microsoft .NET Technologien.

Offensichtlich handelt es sich um zwei völlig unterschiedliche Themenschwerpunkte. So entstanden in beiden Unternehmen auch zwei unterschiedliche Lösungsansätze zur Entwicklung von kundenspezifischer Standardsoftware. Im Dialog wurde jedoch schnell klar, dass keine der beiden Lösungen technologie- oder branchenspezifisch ist. Es ist möglich beide Lösungsansätze mit beliebigen Technologien umzusetzen. Es stellte sich heraus, dass die zentrale Frage nicht die nach dem „Wie“ ist. Die eigentliche Frage ist die nach dem „Was“.

Die Ursachenanalyse

Was also passiert, wenn man zum Beginn der Entwicklung nicht nach dem „Was“ fragt, sondern direkt in Lösungen – also dem „Wie“ denkt? Zwei verschiedene prinzipielle Modelle sind in der Praxis sehr häufig anzutreffen:

Variante 1 – Konfigurationsdateien

Bei diesem Lösungsweg werden Funktionalitäten über Konfigurationsdateien an- und ausgeschaltet. In der Entwicklung erscheint diese Lösung als sehr trivial: Es wird eine Software mit maximalem Funktionsumfang, die sog. 150%-Lösung, implementiert und an den entsprechenden Stellen werden Möglichkeiten zur Deaktivierung integriert. Die eigentliche Komplexität, das Erzeugen und Verwalten der einzelnen Konfigurationsvarianten, wird dabei oft übersehen. Besonders kritisch wird es, wenn neue Features zu der bestehenden Software hinzukommen, da diese bei allen vorhergegangenen Konfigurationen zu berücksichtigen sind. Dennoch kann dies eine praktikable Lösung sein, wenn es sich nur um minimale Anpassungen zwischen den einzelnen Varianten handelt. Komplexe Merge-Vorgänge entfallen, dafür sind jedoch umfangreiche Integrationstests nötig.

Variante 2 - Kundenspezifische Branches

Ausgangspunkt dieser Variante bildet ein initial entwickeltes Softwareprodukt. Dieses wird als Standardlösung definiert und wird auf einem Development-Branch kontinuierlich weiterentwickelt und auf dem Main-Branch synchronisiert. Bei der Anforderung einer kundenspezifischen Lösung wird ein separater Branch für den Kunden von Main abgezweigt. In diesem Kunden-Branch werden alle individuellen Anpassungen durchgeführt. Diese Lösung führt jedoch zu folgenden Problemen:

a) Problembehebungen in der Kundenlösung, welche den Standard-Bereich betreffen, können nicht einfach nach Main gemerged werden. Es ist ein partieller Merge nötig, um die spezifischen Features nicht mit in den Main-Branch zu übernehmen.

b) Funktionen, die nur eine Untergruppe an Kunden betrifft, müssen doppelt entwickelt werden, da sie nur über den Main-Branch ausgetauscht werden können. Dies hätte jedoch zur Folge, dass alle zukünftigen Kunden auch betroffen wären.

Um diese Probleme zu umgehen, werden Branches meist nicht gemerged. Die entsprechenden Code-Zeilen werden per Clone&Own übertragen. Neben der hohen Fehleranfälligkeit geht auch die Traceability irgendwann verloren. Ab einer Größe von etwa 10 Kunden-Branches führt diese Lösung ins Chaos.

In den Softwareprodukten beider Unternehmen wurden die zuvor aufgezeigten Lösungsvarianten umgesetzt. Sie brachten aber nicht den gewünschten Erfolg aufgrund der ebenfalls aufgeführten Nachteile.

Abbildung 1: Der Weg zum nicht beherrschbaren Branch-Model

Es stellen sich nun zwei Fragen:

1. Warum machen zwei Entwicklungsabteilungen (und in Wirklichkeit sind es ja noch viel mehr) scheinbar die gleichen Fehler?
2. Wie sieht die „richtige“ Lösung aus?

Zunächst gilt es festzuhalten, dass es sich keineswegs um grobe Fehler handelt, die hier von den Entwicklern begangen wurden. Primär stehen bei der Entwicklung eines Kundenprojektes die spezifischen Anforderungen im Vordergrund. Kein Kunde wird Ihnen mehr Zeit oder Budget geben, damit Sie ein Framework entwickeln können, das im Anschluss weiteren Kunden verkauft werden kann. Nichtfunktionale Aspekte wie die Modularisierung, Wartbarkeit und Erweiterbarkeit der Software werden dadurch niedriger priorisiert. Hinzu kommt, dass gängige Programmierprinzipien wie YAGNI (You Ain’t Gonna Need It) oder GALAP (Generalize As Late As Possible) das Entwickeln eines Frameworks zum Beginn des Produktlebenszyklus nahezu unterbinden. Es wurden also keine Fehler begangen. Vielmehr zeigt sich hier der Lebenszyklus einer Produktfamilie.

Nun soll die Frage nach der „richtigen“ Lösung beantwortet werden. Aber kann es überhaupt DIE richtige Lösung geben? Die Betrachtung der Unternehmen WITTENSTEIN electronics und AIT zeigt, dass dies wohl nicht der Fall ist. Warum aber sind bei diesen beiden Unternehmen zwei völlig unterschiedliche Konzepte entstanden? Beide lösen doch das Kernproblem, nämlich „Individualisierung und Standardisierung in Einklang zu bringen“. Ein Blick auf die beiden Lösungsansätze wird im Folgenden Klarheit bringen.

Die Produktlinie – Individualisierung durch Dependency-Management

Eine Produktlinie besteht aus mehreren individualisierten Produkten, die eine Reihe von Gemeinsamkeiten aufweisen und dadurch eine zusammenhängende Gruppe bilden. Erst durch die Betrachtung der Aspekte einer Produktlinie konnte die hohe Varianz der Endprodukte bei der WITTENSTEIN electronics ermöglicht werden.

Mit Blick auf die zuvor erläuterten (Fehl-)Ansätze bietet eine Produktlinie weitere wesentliche Vorteile: Die Wiederverwendung von bereits entwickelten Komponenten in Folgeprodukten und die damit verbundene Effizienzsteigerung hinsichtlich der Entwicklungsdauer ist nur ein Beispiel. Der Schlüssel zur Produktlinienentwicklung ist dabei die Ausnutzung der Gemeinsamkeiten zwischen den Produkten der Produktlinie und die Darstellung und Handhabung der unterschiedlichen Produkte. Nachdem die vorherigen Individualisierungskonzepte wenig erfolgreich waren, galt es nun neue Wege zu beschreiten. Hierzu wurden mit Hilfe einer Scoping-Analyse die Bestandteile der ausgelieferten Produkte ermittelt, welche quasi unverändert in allen Produkten vorliegen (Core, Abbildung 2). Dabei wurde ein weiterer Bestandteil ersichtlich, der oftmals denselben Umfang aufweist, aber projektspezifisch individualisiert wurde (Common, Abbildung 2).

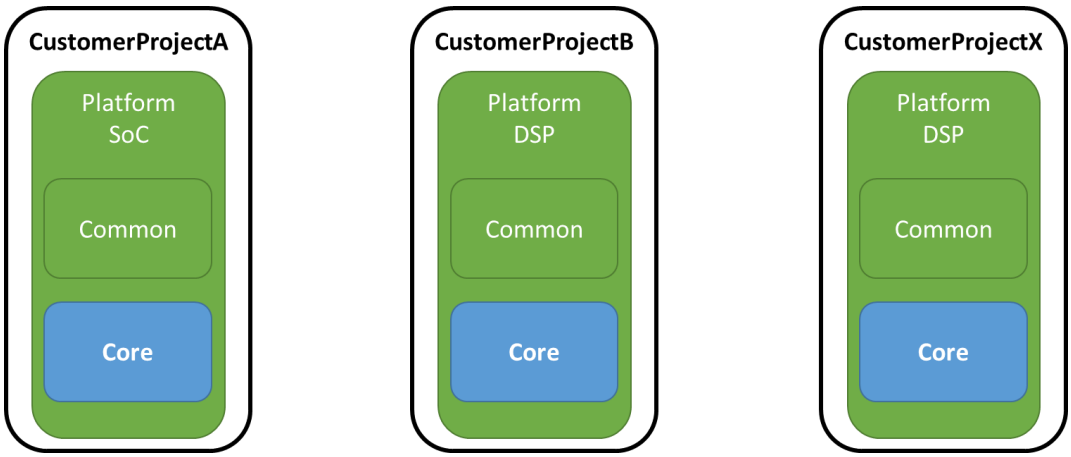

Abbildung 2: Darstellung eines typischen Kundenprojektes nach einer Scoping-Analyse

Diese scheinbar triviale, aber weitreichende Erkenntnis führte zu einem immensen Initialaufwand, der als Ergebnis unter anderem den im Folgenden vorgestellten Ansatz des Dependency-Managements hervorbrachte. Abbildung 3 skizziert diesen erarbeiteten Dependency-Ansatz auf sehr abstrakte Weise, bildet jedoch den Kern der Produktlinienentwicklung.

Um losgelöst vom „Projektgeschäft“ Core-Features entwickeln zu können sowie eine Basis zum Erstellen von neuen Kundenprojekten zu haben, wird neben dem *Core* das *ProjectTemplate* angeboten. Das dargestellte *ProjectTemplate* mit seinen Artefakten *Common* und *Platform* dient als Entwicklungs- und Laufzeitumgebung für den *Core* sowie als Startpunkt für neue Kundenprojekte.

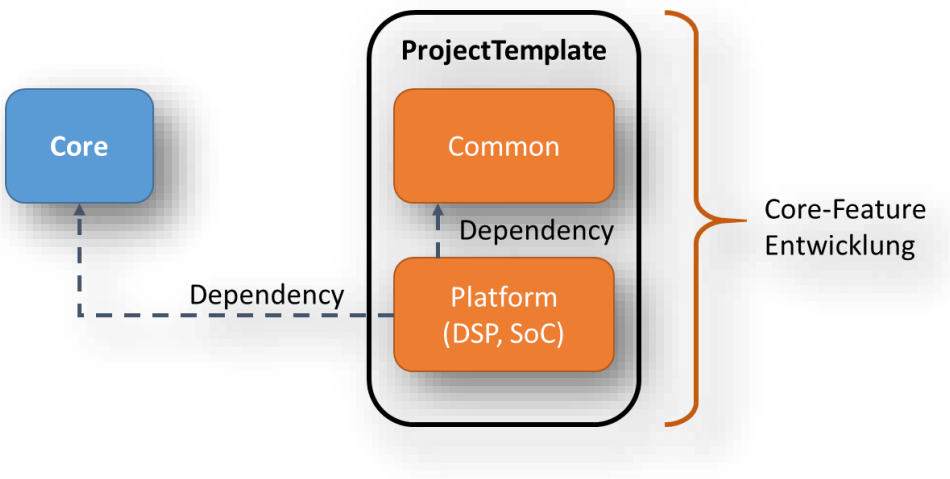

Abbildung 3: Umgebung zur Entwicklung des "Core"

Damit die bis dato erläuterten Artefakte *Core*, *Platform* bspw. *DSP* und *Common* miteinander zu einem lauffähigen Projekt verschmelzen können, kommt der Dependency-Manager der AIT GmbH zum Einsatz. Der Dependency-Manager wird eingesetzt um für jede Plattform, bspw. *DSP*, eindeutige Versionen von *Common* und *Core* zu referenzieren. Hierbei ist *Core* der Teil des Produktes, der projektunabhängig Verwendung findet, also der Wiederverwendungsanteil. *Common* abstrahiert den Teil des Produktes, der kunden- und applikationsspezifisch modifiziert werden kann.

Wie bereits geschildert, hat *ProjectTemplate* polyvalente Eigenschaften:

- Entwicklung von Core-Features
- Aufsetzen von Kundenprojekten

Durch die Implementierung einer wiederverwendbaren Kernkomponente sowie eines individualisierbaren Bestandteils ist es nun sehr einfach, ein neues Kundenprojekt einzurichten. Die generellen Abhängigkeiten zwischen den einzelnen Artefakten *Core*, *ProjectTemplate* und *CustomerProject* sind in nachfolgender Abbildung noch einmal dargestellt.

Die Struktur in *CustomerProject* ist analog zu der Struktur in *ProjectTemplate* aufgebaut. Die beiden Pfeile *Branch* in Abbildung 4 referenzieren von den Release-Branches in *Common* und *Platform* auf die korrespondierenden Develop-Branches in *CustomerProject*. Weiterhin ist die Dependency zum *Core* von *CustomerProject* als „read only“-Dependency ausgeführt, da Core-Features niemals innerhalb *CustomerProject* angepasst werden dürfen. Die Teile des Softwareproduktes in *Common* und *Platform* können nun bei gleichzeitiger Nutzung des standardisierten *Core* individualisiert werden.

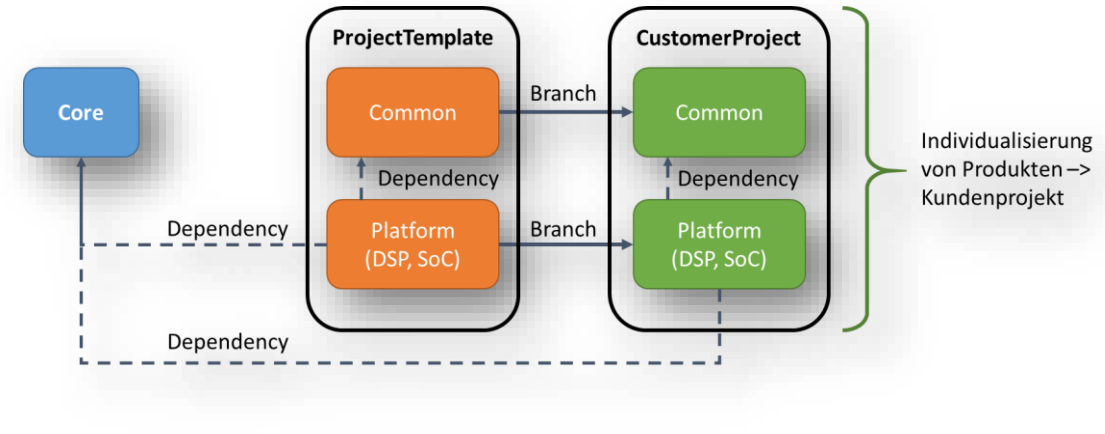

Abbildung 4: Individualisierung eines Kundenprojektes

Mit diesem Ansatz ist es der WITTENSTEIN electronics gelungen, das Paradoxon „kundenspezifische Standardsoftware“ zu lösen, wobei bereits erwähnt wurde, dass das erläuterte Dependency-Management nur ein Pfeiler zur erfolgreichen Umsetzung darstellt.

Individualisierung mittels Paketverwaltung und Extension-Points

Das Konzept einer Produktlinie war bei der WITTENSTEIN electronics der entscheidende Schlüssel zum Erfolg. Dieser Ansatz war bei der AIT GmbH jedoch nicht passend. Vielmehr handelt es sich hier um ein Standard-Produkt, welches in seiner Anzahl an Basis-Funktionen variabel ist und um weitere kundenspezifische Funktionalitäten erweitert wird. Bei den Erweiterungen kann es sich zum einen um neue Features handeln, oder um Anpassungen für bestehende Features. Abbildung 5 zeigt schematisch den Aufbau der Applikation nach der Auflösung der monolithischen Struktur.

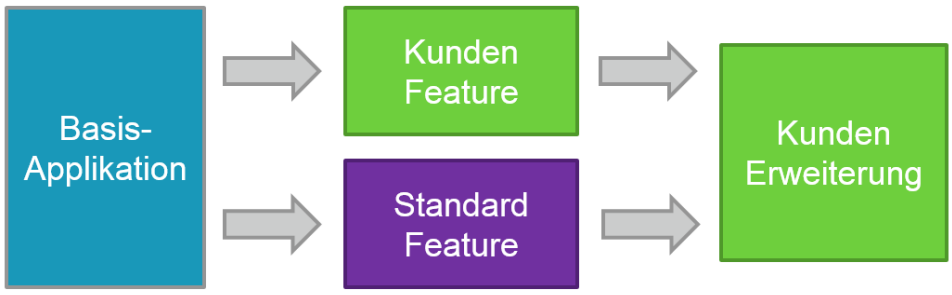

Abbildung 5: Anwendungsstrukturierung

Die Basisapplikation enthält hierbei keine nennenswerten Funktionalitäten. Sie dient lediglich als Hosting-Plattform und wird für jedes Kundenprojekt in einem separaten Branch verwaltet. Durch den minimalen Umfang der Basis-Applikation ist ein Mergen von eventuellen Bugfixes problemlos möglich.

Die Einbindung der Basis-Funktionen erfolgt über sogenannte Pakete. Hierzu wird das Paketverwaltungssystem NuGet eingesetzt. Es ermöglicht das Bereitstellen von

Modulen mit eindeutigen Versionsnummern über eine zentrale Verwaltungsstelle (Netzlaufwerk oder Web-Server). Alle Basis-Features werden daher in eigenen Projekten entwickelt und über einen automatisierten Build-Prozess in der Paketverwaltung veröffentlicht. Von dort können sie in den Kundenprojekten konsumiert werden. Diese Verteilungsart garantiert, dass die Basis-Features in den Kundenprojekten gegen jede Form der Änderung geschützt sind. Da mehrere Versionen eines Features parallel verfügbar sind, obliegt es jedem Kundenprojekt, ob eine Aktualisierung vorgenommen wird oder nicht.

Die Implementierung von kundenspezifischen Features oder gar Anpassungen an bestehenden Features findet über sogenannte Extension-Points statt. Hierbei handelt es sich um fest implementierte Einstiegspunkte in der Basis-Anwendung und den Basis-Features. An diesen Einstiegspunkten suchen die Basis-Anwendung, als auch die Basis-Features nach Modulen mit zuvor definierten, oder konfigurierbaren Signaturen. Wird ein entsprechendes Modul gefunden, werden die spezifischen Funktionen aufgerufen und ausgeführt.

An dieser Stelle stellt sich natürlich sofort die Frage, wo in der Anwendung solche Extension-Points vorzusehen sind und welchen Prinzipien die Erweiterungsmodulen unterliegen um aufgerufen zu werden. Hierauf kann jedoch keine generelle Antwort gefunden werden. Folgt ihre Anwendung jedoch einem iterativen Ablauf, kann zum Beispiel zu Beginn einer jeden Iteration die Suche nach neuen Modulen implementiert werden. Im Anschluss werden alle gefundenen Module in einer zu definierenden Reihenfolge aufgerufen.

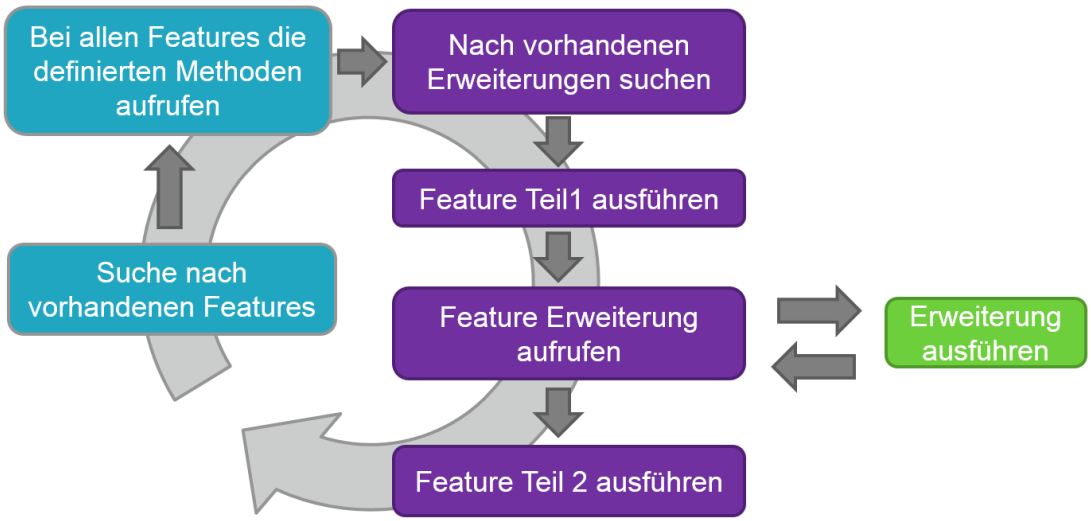

Abbildung 6: Konzeptioneller Programmablauf

Bei der Frage nach dem Design der Module kommt wiederum ein Paket zum Einsatz. Dieses Paket beinhaltet Interface-Definitionen und Template-Dateien, die die Grundlage für die Implementierung einer Erweiterung bilden. Die Entwicklung von kundenspezifischen Funktionen seitens AIT wird natürlich auch erheblich vereinfacht. Die so erstellten Module können nun wiederum als Pakete vor dem

Kompilieren der Software oder auch dynamisch zur Laufzeit durch die Verwendung von entsprechenden Technologien in die Anwendung eingebunden werden.

Wie bereits erwähnt wurde, handelt es sich hierbei um ein Beispiel aus der Applikationsentwicklung im Microsoft-Umfeld. Dies bedeutet jedoch nicht, dass die Grundidee im Umfeld von Embedded-Software nicht auch umsetzbar ist. Es ist durchaus möglich in einer Hauptanwendung sämtliche für ihre Kundenprojekte benötigten Grundfunktionalitäten zu entwickeln. Durch die Verwendung von NuGet können einzelne Features in kleine Pakete verpackt werden. NuGet unterstützt jede Form an Paketinhalten – nicht nur Binaries. Bei der Installation der Pakete in ihrem Zielprojekt wird ein Installationsskript ausgeführt, in dem die für ihr Feature benötigten Dateien entsprechend im Projekt platziert werden. Das Anlegen eines neuen Kundenprojektes könnte demnach folgendermaßen aussehen:

1. Anlegen eines komplett leeren Projektes
2. Installieren des Paketes „Infrastruktur“ zum Anlegen der Ordner-Hierarchie
3. Installieren der gewünschten Feature-Pakete (Diese können auch Abhängigkeiten untereinander haben, welche automatisch aufgelöst werden.)
4. Hinzufügen von neuen kundenspezifischen Funktionen

Durch das versionsbasierte Referenzieren der Pakete ist es nun möglich, einzelne Features gezielt zu aktualisieren. Wird ihr Projekt in einem automatisierten Build-Prozess erstellt, können die Pakete bei jedem Build neu geladen und somit eine Veränderung von Basis-Features ausgeschlossen werden. Hinzu kommt, dass durch die Erstellung der Pakete aus einem zentralen Projekt auf ein aufwändiges Branchmodell für die einzelnen Features verzichtet werden kann.

Fazit

Um individuelle Standardsoftware entwickeln zu können, muss eine zentrale Frage beantwortet werden: Welche Funktionalitäten gehören zum Standard und wo ist eine Individualisierung möglich und nötig? Es wurde gezeigt, wie mit Hilfe von Dependency-Management eine Summe von Standard-Features in eine individualisierte Hauptanwendung integriert werden können. Demgegenüber steht die standardisierte Basisanwendung, die via Paket-Verwaltung und Extension-Points mit Features und Erweiterungen auf Kundenwünsche angepasst werden kann.

Am Ende bleibt die Erkenntnis, dass die Individualisierung von Softwareprodukten eine Herausforderung ist, der man sich stellen muss. Wer die Augen vor diesen Herausforderungen verschließt, wird am Markt nicht bestehen können.

Beide dargestellten Lösungsansätze haben sich in der Praxis bewährt und sind auf Projekte unterschiedlichster Größe skalierbar. Auch wenn der initiale Aufwand für ein passendes Konzept nicht zu unterschätzen ist, so überwiegen die Vorteile klar. Die Entwicklung von kundenspezifischer Software bei gleichzeitiger

Standardisierung zur effektiven Ressourcennutzung stehen somit nicht im Widerspruch zueinander – erst recht nicht mit dem passenden Konzept.

Essig, Matthias

Seit seinem abgeschlossenen Masterstudium in Elektro- und Informationstechnik arbeitet Matthias Essig als Embedded Softwareentwickler bei der WITTENSTEIN electronics GmbH. Seine Themenschwerpunkte liegen im Bereich des Software Engineerings sowie der Software-Projektleitung im Umfeld der elektrischen Antriebstechnik. Weiterhin spielen die Wiederverwendung von Software-Artefakten sowie das damit verbundene Realisieren von kundenspezifischen Softwarelösungen eine zentrale Rolle im Aufgabengebiet von Hrn. Essig. Seit Mai 2015 ist Herr Essig Leiter der Embedded Software Entwicklung bei der WITTENSTEIN electronics GmbH.

Boost, Benjamin

Benjamin Boost ist Senior Consultant bei der AIT GmbH & Co. KG. Er ist seit 5 Jahren als Software-Entwickler und Architekt im industriellen Umfeld tätig. Sein Themenschwerpunkt liegt auf der Konzeptionierung und Entwicklung von verteilten Anwendungen auf der Basis von Microsoft Technologien. Darüber hinaus unterstützt er Unternehmen bei der Einführung und Anpassung des Team Foundation Servers.