embedded Clean code im A-SIL-Serienentwicklungsumfeld

Praktische Erfahrungen hochwertiger Softwareentwicklung

Thomas Winz, softwareinmotion GmbH

Jurassic Park [R1]: „Sie haben befürchtet, Tiere zu verlieren, und das Programm ist deshalb so ausgelegt, daß es sofort Alarm schlägt, wenn es weniger als die erwartete Anzahl sind. Aber das ist gar nicht das Problem. Das bei weitem größere Problem ist, daß Sie mehr als die erwartete Anzahl haben.“ Wer kennt nicht unbedachte systementscheidende Anforderungen?

Das zugrundeliegende Referenzprojekt ist die erste selbständige inhouse Produktentwicklung eines ASIL Steuergerätes der softwareinmotion. Als gleichwertige Stakeholder wurden die Vertriebspartner und externen ISO Assessoren aufgestellt.

Die Produktentwicklung erfolgte nach Scrum. Unter dem Begriff „additiver Architekturstil“ wurden die umfangreichen agilen Tailoringsmaßnahmen der ISO 26262 zusammengefasst, u. a.: „Anforderungen als Test“, Risikoarchitektur, TDD, Code Hygiene, continuous Test, Deploymentchain und agile HW. Die notwendige Güte des Quellcodes wurde durch embedded Clean code gewährleistet. Zusätzlich musste Infrastruktur geschaffen und ISO Safetyargumente beschrieben werden.
Da alle Tätigkeiten nur der Produktbacklogpriorisierung folgen, ermöglicht unsere agile Einstellung ein erfolgreiches Projekt. Wir konnten unseren extrem unsicheren Projektalltag als lebenswerten und planbaren Raum gestalten.

Extreme Unsicherheiten

Mit der „lean Startup/Startup Way“ Methode stellte Eric Ries fest, dass die Arbeit am Produkt in einer Entwicklungsabteilung „unter extremen Unsicherheiten“ leidet. [R2] Das Verfahren der kontinuierlichen Innovation (Bild 1) besagt, dass Aufwände nur erbracht werden, wenn diese im Projekt die allgemeine „Unsicherheit“ verringern. Das klassische V-Modell wird erweitert. Diese Bewertungsmethode für Erfolg bedeutet einen fundamentalen Unterschied zu „agile aber“- oder klassischen Projektbewertungsmethoden.

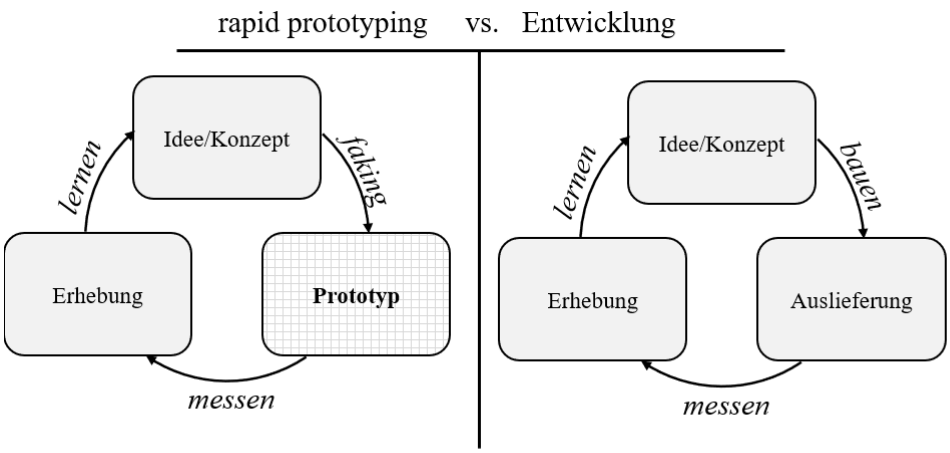


Bild 1: Kontinuierliche Innovation

Mentale Modelle [R3] hochqualifizierter Spezialisten kompensierten Projektunsicherheiten, da bei embedded Produkten technische Anforderungen dominieren. Moderne Konsumerbenutzererlebnisse lassen aber im industriellen Umfeld die Wichtigkeit von fachlichen Anforderungen steigen (siehe Tabelle 1). Damit wächst das unternehmerische Risiko für technisch versteifte Unternehmen gegenüber agilen Konkurrenten.

/	Anforderung technische	fachliche
Voraussetzungen	*der Technik*	*vom Kunden*
Beispiel	• *Regelstrecke* • *Versorgungsspannung* • *Speichergröße*	• *Benutzbarkeit* • *Data Security* • *Diagnose (UDS)*
Zusammenhang		

Tabelle 1: Anforderungsarten

Prototypen sind funktionale Hilfsmuster, die vor Entwicklungsbeginn genutzt werden, um Unsicherheiten zu klären (siehe Bild 1). Bei der Validierung täuschen diese Platzhalter das Produktverhalten vor. Im hochsicherheitskritischen Umfeld beginnt der Lebenszyklus eines ASIL Produktes als abstraktes Systemmodell der Sicherheitsanalyse. Der Prozess SPRINT beschreibt, wie projektkritische Unsicherheiten innerhalb von fünf Tagen beantwortet werden. [R5]
Im Referenzprojekt wurden Prototypen intensiv genutzt, beispielsweise: Excel, VBA Skripte, Arduino Sketch, Dev. Kits, QT und Paint. Die Formulierung einer soliden Frage war dabei die größte Herausforderung, gefolgt von verständlicher Dokumentation. Unklarheiten in der Fragestellung führten zu aufwendigen Prototypen, die dadurch in der verfügbaren Zeit keine konkrete Antwort lieferten.

Agile HW Entwicklung
HW Entwicklungswerkzeuge haben bis vor wenigen Jahren einen agilen Ansatz in der HW Entwicklung verhindert. Im Allgemeinen lassen sich SW Unsicherheiten/Anforderungen im embedded Bereich ohne HW nicht klären. Die als „Big Bang“ bekannte Methodik, die erst spät im Projekt verschiedene Software- und Hardwareelemente in einem großen Schritt zu einer Komponente oder einem Gesamtsystem integriert, gilt als veraltet.

Dr. Tobias Kästner und Mario Bunk [R6] haben mit ihrem „agilen HW-Entwicklungskonzept“ gezeigt, dass jegliches HW-Feature vorhersagbar und in wenigen Wochen umsetzbar ist. Dazu wird jedes Feature auf einzelne Featureplatinen ausgelagert und in einen projektspezifischen Entwicklungsrig integriert. Das Hinzufügen, der Austausch und das Refactoring einzelner HW Features sind somit nur noch von der Produktbacklogpriorisierung abhängig.

Bild 2 zeigt den selbst entwickelten Universalteststand. Dieser Teststand nutzt die Entwicklersoftwarewerkzeugkette, um notwendige Fahrzeugeigenschaften wie Bordnetz, K15 Manipulation und weiteres zu emulieren. Auf dieser Basisinfrastruktur setzt der eigenentwickelte Remotelaborplatz (Bild 2) auf. Der Remotelaborplatz bietet die Möglichkeit das „Gerät unter Entwicklung“ gezielt zu manipulieren. Ein PicoScope rundet die Telemetrie-Möglichkeiten ab.

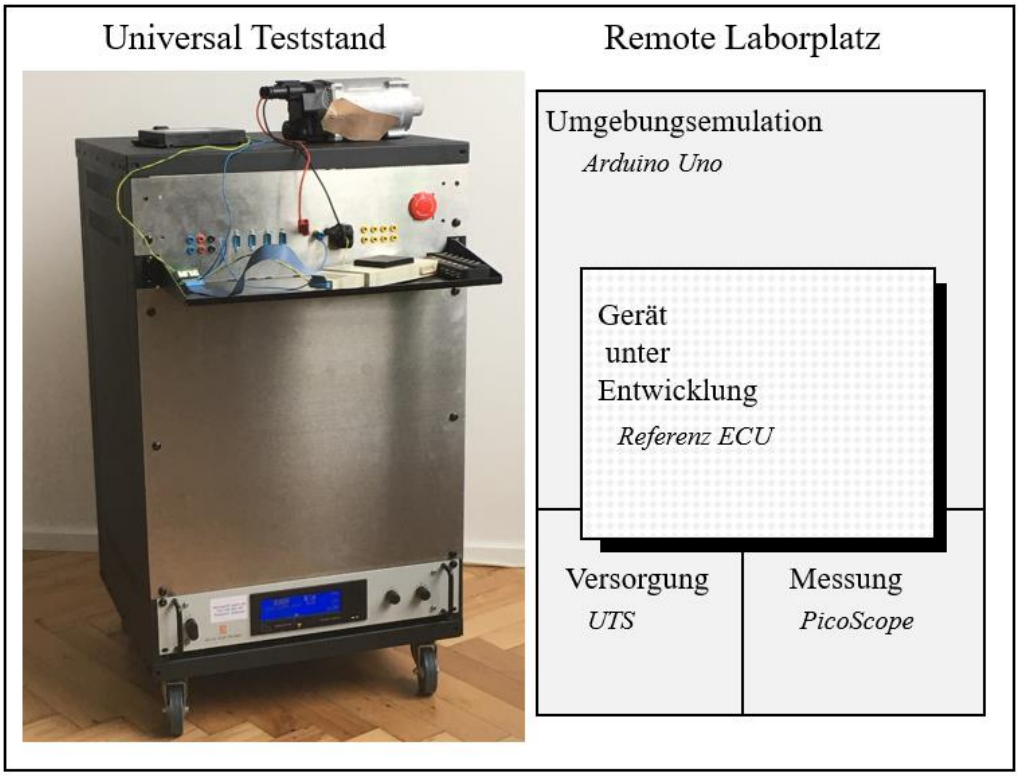


Bild 2 Infrastrukturlösungen

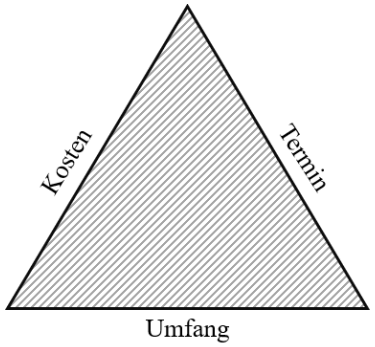
Im Referenzprojekt ermöglichen diese Infrastrukturlösungen Tätigkeiten, die Entwickler im Alltag durchführen, u. a.: Langzeittests, Mock von HW-Funktionen. So wurde die Abstimmung des HW-SW-Interface wesentlich vereinfacht. Mit Hilfe eines einfachen Arduino Sketch erfolgte die Inbetriebnahme der Referenzfeatureplatine. Somit war bereits vor der Entwicklung der SW-Komponente „Hardwareabstractionlayer“ das Verhalten der HW bekannt. Dies führte zu weniger Aufwand in der Entwicklung des darauf aufbauenden SW-Stack. Fehlende Features der HW konnten frühzeitig identifiziert und bei der Aktualisierung der Referenzfeatureplatine berücksichtig werden.

Architektur

Alle systemrelevanten Entscheidungen über das Projekt sind in der Architektur zu treffen. Der Architekt hat die Aufgabe, die Erwartungen der Stakeholder an das Projekt so weit wie möglich zu erfüllen. [R7] Zusammen mit dem Productowner/Projektleitung werden mit Hilfe des Modells „Projektumfeld“ (Bild 3) die Priorisierung im Produktbacklog (Projektplanung) vorgenommen.

Agile Projekte unterscheiden sich auch hier wesentlich zu „agile aber“- oder klassischen Projekten. Einen Grund hat Debbie Madden 2014 genannt: „Ein echtes Agile Software-Entwicklungsprojekt kann nur eine Dreieckskante festgelegen. Die beiden anderen Kanten müssen flexibel bleiben. Ansonsten handelt es sich nicht um

ein Projekt, das sich mit der agilen Entwicklung umsetzten lässt.“ [R8] Die Projektplanung kann also immer nur eine Kante festlegen und muss den Umfang der zwei verbleibenden Kanten kontinuierlich mit den Stakeholdern klären.



Ergeben sich aus den Antworten auf:
- Was ist das Ziel der Projektphase?
- Wer ist fachlich involviert?
- Wer ist leitend involviert?
- Wer ist strategisch involviert?
- Wer stellt Regeln auf?
- …

Bild 3: Projektumfeld

Im Referenzprojekt wurde diese Einstellung mit dem additiven SW-Architekturstil gelebt. Kontinuierliche Refactoringswartungsarbeiten ermöglichten ein Hinzufügen von fachlichen Anforderungen als zusammenhängende Funktionsmuster. Dabei sind die systemrelevanten Aufwände der Funktionsmusterintegration über die Projektlaufzeit konstant zu halten. Beispiel: Erweiterung der Diagnose um einen Selbsttest zieht kein Rewrite vom Kommunikationstack nach sich. Hinzufügen, Austausch und Refactoring einzelner SW Features sind somit nur noch von der Produktbacklogpriorisierung abhängig.

Entwicklung

Im Referenzprojekt werden zwei unabhängige Werkzeugketten genutzt (Bild 4). SW-Werkzeuge der oberen Werkzeugkette sind in der automatischen Jenkins-Deploypipeline eingebunden. Die gleiche Instanz von Jenkins wird zudem für die kontinuierliche Testpipeline genutzt.

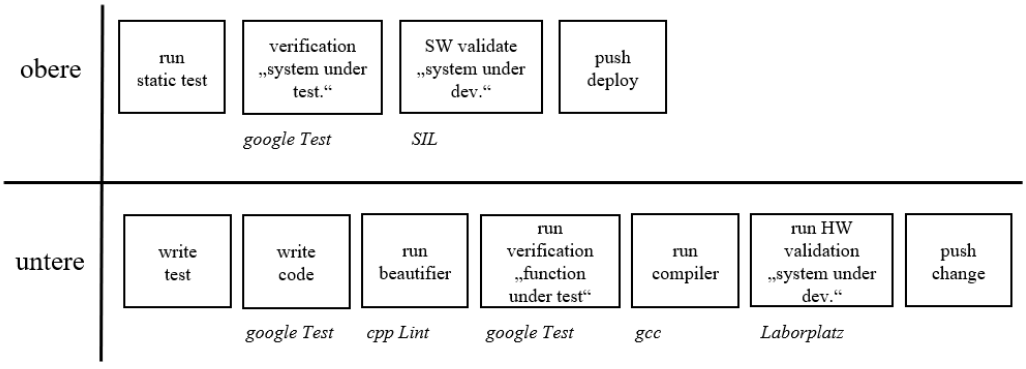


Bild 4 Entwicklungswerkzeugketten

Die obere Werkzeugkette besteht aus normqualifizierbaren SW-Werkzeugen, die für Freigabeaufwände von Auslieferungen genutzt werden. Die Vereinfachung der ISO 26262 Werkzeugqualifikation setzt voraus, dass die obere Werkzeugkette nicht auf Produkte der unteren Werkzeugkette aufbaut. SW-Werkzeuge der unteren Werkzeugkette sollten den ISO 26262 tool confidence level von TCL 1 besitzen und sind somit von einer Werkzeugqualifikation ausgenommen. [R9]

Durch diese Trennung werden untere SW-Werkzeuge nur noch nach ihrem Einfluss auf die Entwicklungsumgebung ausgewählt. Somit konnten Methoden wie TDD und Code Hygiene die Synergien aktueller Open-Source-Projekte nutzen.

Um dem Dokumentationsaufwand sicherheitskritischer Produkte gerecht zu werden, werden alle Architekturinformationen mit dem Werkzeug „Doxygen“ aggregiert. Die Dokumentation, welche nicht in Sourcecodetags erfolgen kann, wurde in Markdown Dateien ausgelagert. Um auch Anforderungen durch diese Methode zu dokumentieren, muss ein strenger Absichtsnachweis(Verlinkung) erfolgen.

Auslieferung

Der additive SW-Architekturstil setzt gleichbleibende Aufwände der Funktionsmusterintegration über die Projektlaufzeit voraus. Carola Lilienthal stellte fest, dass Änderungen im Sourcecode die technische Schuld erhöhen (Bild 5). [R10]

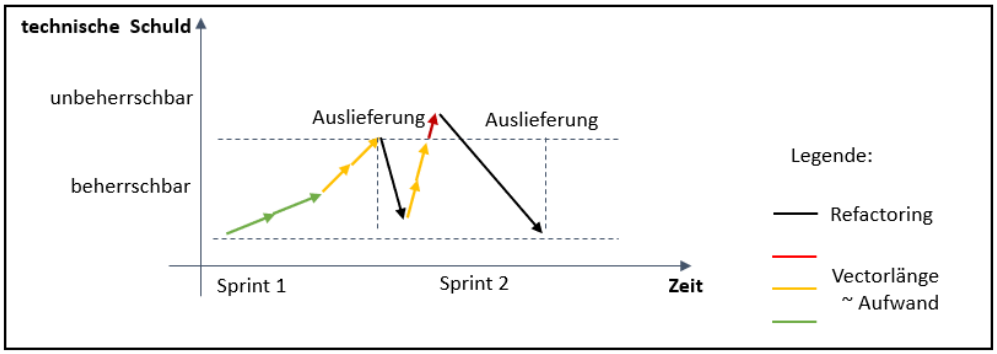


Bild 5 Technische Schuldentwicklung [R10]

Technische Schulden sind alle Aufwände, die nicht zu den notwendigen Aufwänden der Umsetzung zählen. Die statische Analyse misst diese durch verschiedenste Metriken. Tabelle 2 zeigt die verschiedenen Verursacher technischer Schuld auf. Nach der Analyse leiten sich die verschiedenen Refactoringmaßnahmen ab. Im Unterschied zum inkrementellen Wasserfall-/V-Modell [R11] werden Refactoringwartungsmaßnahmen technischer Anforderungen erwartet.

/	Zertifizierung	Architektur	Entwicklung
Ursache	• *Anforderungen der ISO nicht erfüllt*	• *Architekturstil verletzt* • *Architektur dient Quellcode nicht mehr*	• *Code Hygiene nicht eingehalten*
Beispiele	• *Anforderungsabdeckung* • *schwaches Argument*	• *wachsende Schnittstelle* • *Durchbrüche*	• *zu komplexe Lösung*

Tabelle 2: Technische Schulden

Im Referenzprojekt wurde zum Auslieferungszeitpunkt die technische Schuld ermittelt. Der Architekt, Lead Dev und Integrator bewerteten rollenrelevante technischen Schulden. Das Ziel vom gleichbleibenden Integrationsaufwand wird

gelebt. Notwendige Refactoringmaßnahmen waren somit nur noch von der Produktbacklogpriorisierung abhängig.

Quellenverzeichnis

R	Titel	Autor/Link
1	Dino Park	Michael Crichton
2	The Startup Way, lean Startup	http://www.thestartupway.com http://theleanstartup.com
3	Smarter Faster Better,	Charles Duhigg, Kapitel 3: „3. Fokus“
4	Agile in Automotive	http://www.euroforum.de/agile-automotive/
5	SPRINT	http://www.gv.com/sprint/ https://www.amazon.de/Sprint-Tagen-Ideen-testet-Probleme/dp/3868816380
6	SQ Magazin Ausgabe 44, Agilität	https://www.asqf.de/
7	Stakeholdererwartungen	https://de.wikipedia.org/wiki/Projektmanagement#Stakeholdererwartungen
8	agile Contract	https://www.stridenyc.com/blog/im-agile-but-my-contract-isnt-how-to-align-contracts-with-agile-software-development-teams/ Zitat bei Übersetzung angepasst
9	Confidence in the use of software tools	ISO 26262-8:2016[E] 11.2; 11.4.1; 11.4 .6.1
10	Langlebige Software-Architekturen	Carola Lilienthal
11	Wasserfallmodell	https://de.wikipedia.org/wiki/Wasserfallmodell

Autor

2010 hat Thomas Winz an der HTWG Konstanz den Studiengang „Technische Informatik“ als Bachelor of Sience erfolgreich abgeschlossen. Seitdem ist Thomas Winz Softwareentwickler im Automotive-Bereich. Nach 5 Jahren im Projektauftrag bei verschiedensten Kundenentwicklungen ist Thomas Winz "lead programmer" im ersten Automotive inhouse Projekt. Weitere Vorträge vom Referenten: 2014: Embedded Clean Code - Weltpremiere (Vortrag) 2015: Embedded Clean Code - sichert Qualität und Effizienz (Vortrag)