

Auswahl einer ARM Entwicklungsumgebung orientiert an Projektanforderungen



Gutes Werkzeug ist halbe Arbeit

Für erfolgreiches Software Engineering sind die Werkzeuge, die den Softwareentwicklern zur Verfügung stehen mit entscheidend. Manche Microkontroller-Familien lassen an dieser Stelle keine große Wahl zu und Entwickler sind froh, wenn die einzigen verfügbaren Werkzeuge halbwegs stabil funktionieren.

Anders bei Microkontroller-Familien (Architekturen), die sich breit am Markt durchgesetzt haben wie z.B. die 8051, C166 oder die ARM-Architekturen. Hier steht eine breite Auswahl an Werkzeugen zur Verfügung.

Grundsätzlich eine hervorragende Situation für effizientes Software- Engineering, aber

wird sie auch genutzt? Dass ein C-Compiler benötigt wird steht nicht zur Diskussion. Aber schon bei der Entscheidung ein RTOS einzusetzen scheinen die Vor- und Nachteile nicht so recht klar zu sein. Und was hat es mit OOP auf sich? Sollte bei einem neuen Projekt lieber in C++ programmiert werden? Zu häufig werden derartige Entscheidungen aus Zeitmangel von einem Projekt zum nächsten verschoben und das Software-Engineering bleibt auf der Strecke.

Das es sich lohnt, sowohl aus finanzieller Sicht, als auch aus Sicht der Effizienzsteigerung, sich mit den Anforderungen auseinander zu setzen und darauf basierend die Tools auszuwählen, möchten wir exemplarisch in der Zusammenstellung von 4 verschiedenen Entwicklungsumgebungen für die ARM Architektur zeigen.

Immer stehen die Anforderungen der Projekte im Vordergrund. Anforderungen wie Lebensdauer, Wartbarkeit und Erweiterbarkeit der Produkte, aber auch Sicherheit und Robustheit der zu entwickelnden Software. Und nicht vergessen sollte man die Erfahrungen der Entwickler im Software- Engineering, als auch die Einarbeitungszeiten in neue Techniken und dem gegenüberstehende ‚Time To Market‘ Situationen.

Ebenso wurde bei der Zusammenstellung darauf geachtet, dass die Werkzeuge aufeinander abgestimmt sind und sich im Idealfall in diesen oder ähnlichen Kombinationen bereits in Projekten bewährt haben.

Ein weiterer Punkt auf den geachtet wurde ist, dass die Entwicklungsumgebungen ohne Kompatibilitätsprobleme, an steigenden Anforderungen orientiert, erweitert werden können. So, dass der Return of Invest in die Tools und in die Einarbeitungszeiten über einen langen Zeitraum gesichert ist, auch wenn sich Engineering-Anforderungen ändern. Ebenso bietet die Erweiterungsmöglichkeit eine ideale Voraussetzung für einen schrittweisen Einstieg in neue Technologien und damit die Verteilung von Investitionen und Einarbeitungszeiten auf einen längeren Zeitraum.

Variante 1

Diese Variante basiert auf folgenden Anforderungen eines Kunden:

Hardware Architektur: Aus Kosten und EMV-Gründen ist die Architektur als Single Chip (kein externer Speicher) geplant.

Echtzeitanforderungen: sind hoch, es gibt jedoch nur wenig Nebenläufigkeiten.

Lebensdauer des Produktes: sind 15 Jahre, wobei umfangreiche Änderungen nicht zu erwarten sind. Wartbarkeit muss über den ganzen Lebenszyklus gesichert sein.

Time To Market: es stehen nur wenige Monate zur Realisierung zur Verfügung.

Qualität: Änderungen des Produktes nach der Auslieferung verursachen hohe Kosten im Bereich Service und sind zu vermeiden.

Erfahrung: in der Anwendung der Hochsprache ANSI-C auf verschiedenen Hardwareplattformen ist vorhanden, jedoch keine Erfahrung mit der ARM-Architektur. Erfahrungen im Einsatz eines RTOS oder anderen Werkzeugen zum Architekturdesign sind nicht vorhanden.

Budget: es steht ein kleines Budget für die Anschaffung von Entwicklungstools zur Verfügung.

Auswahl der Entwicklungsumgebung orientiert an den obigen Anforderungen

Da feststeht, dass das Projekt auf Basis eines Single-Chip-Designs niemals mehr als 256 K-ROM benötigt, wird die kostengünstige Keil RV ARM Compiler Version mit Speicher-Einschränkungen als Basis der Entwicklungs-Umgebung ausgewählt, um mehr Gestaltungsspielraum mit dem geringen Budget zu bekommen.

Die Analyse des Architektur-Designs ergibt nur wenige Tasks mit unterschiedlichen Nebenläufigkeiten. Trotz der geplanten langen Lebenszeit ist die Wahrscheinlichkeit gering, dass komplexe Erweiterungen zu erwarten sind. Diese Umstände lassen auf eine einfache Laufzeitarchitektur schließen, die sehr effizient auf Basis einer main()-Loop realisiert werden kann. Aus diesem Grund und zugunsten der Einarbeitungszeit wird auf den Einsatz eines RTOS verzichtet. Echtzeitanteile mit definierten Reaktionszeiten werden auf der Interrupt-Ebene realisiert.

Die Software-Entwicklung mit dieser Umgebung erfolgt in gewohnter Manier: manuelle Erstellung des C-Codes. Trotzdem wird auf handwerklich korrekten ANSI C-Code Wert gelegt. Die Anforderungen an die Code-Qualität werden durch die Integration eines statischen Code- Analysers (PC-Lint) in die Entwicklungsumgebung sichergestellt.

Die geforderte Robustheit des Produktes zur Auslieferung und wenig Weiterentwicklung haben zu der Entscheidung geführt, die Tests händisch durchzuführen. Dieses geschieht auf Basis der Keil µVision Debugging Umgebung und dem U-Link Debugging Interface.

Die Anforderung an lange Lebensdauer und evtl. notwendige kleinere Anpassungen erfordert eine gute Dokumentation, um sich auch nach Jahren wieder effizient und sicher in die Software einarbeiten zu können. Um dieser Anforderung gerecht zu werden, jedoch gleichzeitig den Aufwand für die Dokumentation so gering wie möglich zu halten und gleichzeitig Inkonsistenz zwischen Code und Dokumentation vorzubeugen, haben wir uns für den Einsatz des Dokumentation Tools Doxygen entschieden.

Auf Grund des hohen Zeitdruckes und dem Umstand, dass noch keine Erfahrungen im Einsatz der ARM-Architektur vorhanden sind, wird eine 3-tägige ARM-Schulung eingeplant, um lange Einarbeitungszeiten in die HW-Architektur zu vermeiden.

In der Summe zeichnet sich die ausgewählte Umgebung durch geringen Einarbeitungsaufwand aus. Es werden, abgesehen von der ARM-HW, keine unbekanntes Technologien eingesetzt und daher wenig unerwünschte Verzögerungen in der Realisierung erwartet.

Trotz des geringen Budgets für die Entwicklungsumgebung, wurde auf Basis der preiswerten limitierten Version des Compilers (die in diesem Fall keine Einschränkungen beinhaltet) ,ein ARM- Seminar und die Anschaffung von PC-Lint ermöglicht.

Bestandteile

- Keil RV ARM Compiler mit Speicher-Einschränkungen
- PC-Lint
- Doxygen ⁽¹⁾
- U-Link
- Eva Board
- ARM-Schulung (3 Tage Basis)

Gesamtkosten 4.000 EUR

⁽¹⁾ Doxygen ist nicht im Lieferumfang enthalten. Nutzung als GNU General Public License

Ergänzend zu obiger Umgebung raten wir zum Einsatz eines Configuration Management Tools.

Variante 2

Diese Variante basiert auf folgenden Anforderungen eines Kunden:

Hardware Architektur: die Stückzahl Produktion der Applikation liegt bei ca. 10.000 Stück pro Jahr bei einem Produktpreis von ca. 400,- €. Aus Kosten und EMV-Gründen ist die Architektur als Single Chip (kein externer Speicher) geplant. Auf Grund der komplexen Bedienung und Mehrsprachigkeit des Systems und einer daraus resultierenden Speicheranforderung für die Texte steht noch nicht fest, ob bei diesem Konzept geblieben werden kann oder doch externer Speicher benötigt wird.

Echtzeitanforderungen: sind lediglich in zwei Nebenläufigkeiten hoch, es gibt jedoch weitere Nebenläufigkeiten u.a. ein komplexes GUI und mehrere Kommunikationskanäle, die als Bus realisiert werden müssen (darunter CAN und USB).

Lebensdauer des Produktes: sind 7 Jahre. Erfahrungsgemäß steigen die Anforderungen in dieser Zeit und es wird Weiterentwicklungen geben. Wartbarkeit und Erweiterbarkeit muss über den ganzen Lebenszyklus gesichert sein. Die Erfahrung der Vergangenheit hat gezeigt, dass auch nach der Produkteinführung die Entwicklung kontinuierlich weiter geht und neue Anforderungen umgesetzt werden müssen. Dieses ist in besonderem Maße zu gewährleisten.

Time To Market: Bis zur geplanten Markteinführung sind es noch 11 Monate. Es stehen zwei Software-Entwickler zur Verfügung.

Qualität: Ausfälle des Systems sind nicht dramatisch und können nachgebessert werden. Trotzdem sind sie zu vermeiden, da der Ruf des Unternehmens leiden könnte.

Erfahrung: in der Anwendung der Hochsprache ANSI-C auf verschiedenen Hardwareplattformen ist vorhanden, jedoch keine Erfahrung mit der ARM-Architektur. Erfahrungen im Einsatz eines RTOS oder anderen Werkzeugen zum Architekturdesign sind nicht vorhanden. Als Configuration Management Tool ist Microsoft Visual SourceSafe im Einsatz.

Budget: Die Erfahrungen der letzten Zeit haben gezeigt, dass mit der vorherigen Vorgehensweise die Komplexität und Qualität nicht dem gewünschten Ziel entsprachen. Es steht ein Budget für die Anschaffung der üblichen Entwicklungstools (Compiler Debugger Eva Board) zur Verfügung. Darüber hinaus kann das Management von Investitionen für notwendige Software Engineering-Maßnahmen überzeugt werden, wenn sie im Rahmen liegen und deren Einführung die Entwicklungszeit nicht zu stark belasten.

Auswahl der Entwicklungsumgebung orientiert an den obigen Anforderungen

Die Umgebung ist für die Entwicklung von Applikationen eines mittleren Komplexitätsgrades gedacht.

Die Wiederverwendung und Wartbarkeit der Software werden durch die Integration eines RTOS verbessert, das Unterstützung beim Design der Laufzeitarchitektur bereitstellt. Außerdem vereinfacht das RTOS die Schnittstellen zu einem gut entkoppelten Hardware Abstraction Layer über asynchrone Kommunikations-Mechanismen.

Die Wahl fiel in diesem Fall auf die Keil/ARM RealView Real-Time Library. Sie beinhaltet neben einem RTOS auch ein CAN und USB Device Interface.

Diese Umgebung stellt einen Mittelweg zwischen der reinen Programmierung auf C-Ebene und der Modellierung auf UML-Ebene dar. Das RTOS liefert wertvolle Mechanismen für das Architekturdesign, während weiterhin wie gewohnt in C programmiert wird.

Die Einarbeitungszeit in ein RTOS liegt bei wenigen Tagen. Diese kann oft bereits durch die Erstellung eines Architekturdesigns im maßgeschneiderten RTOS Start Up Workshop wieder eingeholt werden. Auf Grund der sehr kurzen Entwicklungszeit wurde auch eine Schulung für die Nutzung der CAN und USB Treiber eingeplant. Gleichzeitig wurde in Erwägung gezogen die Realisierung der CAN und USB als Auftragsentwicklung extern zu vergeben, wenn der Zeitpunkt für die Produkteinführung gefährdet wird.

Für das effiziente Design des GUI wurde der Embedded-Display Builder als Designhilfe eingesetzt.

Bestandteile

- Keil Compiler IDE
- Keil Realtime Lib
- Evaluation Board
- U-Link
- Embedded-Display Builder
- ARM-Schulung (Offene Schulung)
- RTOS Start Up Workshop (In House)
- ARM Real-Time Lib Workshop (In House)

Gesamtkosten ca. 16.500 EUR

Eine vergleichbare Lösung ist auch auf Basis der IAR ARM in Kombination mit embOS als Laufzeitumgebung denkbar.

Variante 3

Diese Variante basiert auf folgenden Anforderungen eines Kunden:

Hardware Architektur: das geplante Projekt wird in einer mittelgroßen Stückzahl (ca. 10.000 pro Jahr) produziert werden. Der Endpreis liegt bei ca. 2.500,- € pro Gerät, damit fallen die Kosten für die Rechner-Architektur mit ins Gewicht. Trotzdem hat sich das Produkt-Management dazu entschieden nicht an Speicher und Rechenleistung zu sparen.

Laufzeitarchitektur Anforderungen: in wenigen Nebenläufigkeiten gibt es hohe Echtzeit-Anforderungen. Darüber hinaus gibt es weitere Nebenläufigkeiten, die jedoch mit Reaktionszeiten im 100 ms Bereich recht geringe Zeitanforderungen haben. Es wird mehrere Varianten des Produktes geben und innerhalb des Produktes noch einmal mehrere Betriebsmodi. (Service- und Diagnose-Modus, Kalibrierungs-Modus, StandBy-Modus, Daten Abgleichs- und Lade- Modus ...) Durch die daraus resultierenden Abhängigkeiten und Sonderzustände entsteht aus Erfahrung eine große Komplexität.

Lebensdauer des Produktes: sind ca. 8 Jahre. Erfahrungsgemäß steigen die Anforderungen in dieser Zeit kontinuierlich. Weiterentwicklungen müssen auf Basis von ausgelieferten Produkten geschehen und Upgrade-fähig sein. Wartbarkeit und Erweiterbarkeit muss über den ganzen Lebenszyklus gesichert sein.

Standards: Einige Varianten des Produktes werden als Halbzeug gefertigt, und von verschiedenen Endkunden in deren Produkte integriert. Einige der Kunden fordern die Einhaltung des MISRA Standards. Schnittstellen (z.B: CAN Open) müssen definiert und eingehalten werden.

Time To Market: Bis zur geplanten Markteinführung sind es noch 12 Monate. Es stehen 4 Entwickler im Software-Engineering zur Verfügung. Bisher hat es immer wieder Änderungen in letzter Minute gegeben, weil Bedienkonzept oder andere Dinge nicht wie erwartet waren. Sorgfältig erstellte Pflichtenhefte konnten die Anzahl der Change Requests in der Vergangenheit nicht senken.

Qualität: Ausfälle des Systems können zu Image-Verlust führen. Die Firma ist derzeit bekannt durch die Robustheit der Systeme, vor allem auch in der Bedienphilosophie der Geräte.

Erfahrung: in der Anwendung der Hochsprache ANSI-C auf verschiedenen Hardwareplattformen ist vorhanden, jedoch keine Erfahrung mit der ARM-Architektur. Erfahrungen im Einsatz eines RTOS oder anderen Werkzeugen zum Architekturdesign sind teilweise auf Basis des RTOS embOS vorhanden.

Budget: es steht ein Budget für die Anschaffung von Entwicklungstools zur Verfügung. Darüber hinaus kann das Management von Investitionen für notwendige Software Engineering Maßnahmen überzeugt werden, wenn Sie Qualität und Liefertreue sichern.

Auswahl der Entwicklungsumgebung orientiert an den obigen Anforderungen

Orientiert an der Lebensdauer von 8 Jahren, der gewünschten hohen Qualität und den besonderen Anforderungen an Variantenreichtum und Modi der Applikation, wird mit einer großen Komplexität des Architektur-Designs gerechnet. Um Wartbarkeit und Änderbarkeit über den gewünschten langen Zeitraum trotzdem sicher zu stellen, wird die UML als grafische Modellierungssprache eingesetzt.

Die UML mit einem Modellierungstool ermöglicht die schnelle Modellierung von Prototypen. Auf diese Weise sollen die Tests der Bedienphilosophie früher durchgeführt werden, um mögliche Änderungswünsche so früh wie möglich zu bekommen.

Mit Hinblick auf die Zukunft wird auf Standards wie MISRA, UML und einen HAL (Hardware Abstraction Layer) gesetzt. Gleichzeitig nähern wir uns mit der folgenden Umgebung Standards, deren Einhaltung möglicherweise in den kommenden Jahren gefordert werden (z.B. AUTOSAR, SPICE, IEC61508).

UML

Auf Grund der vielen Betriebsmodi und Nebenläufigkeiten konzentriert sich das Software- Engineering auf die Architekturebene. Das Architekturdesign wird mit der Notation UML modelliert, welche sich auch im Embedded-Bereich als standardisierte Modellierungssprache immer weiter verbreitet.

Die Nutzung von UML bietet eine Reihe von Vorteilen gegenüber der herkömmlichen Software-Entwicklung:

- Langfristige Verstehbarkeit und Wartbarkeit der generierten Software
- Verbesserung der Projektdokumentation, Verständlichkeit auch für Projektfremde
- Konsistenz von Modell, Code und Dokumentation
- Einfache Wiederverwendung von Komponenten in verschiedenen Varianten durch Objekt Orientierte Kapselung.
- Klare Definitionen von Schnittstellen zu „tiefer“ gelegenen Teilen einer Applikation, z.B. Hardware Abstraction Layer

Varianten

Um die unterschiedlichen zukünftig geplanten Varianten und die unabhängige Wartung und Weiterentwicklung möglichst effizient zu gewährleisten wird der Einsatz eines Konfiguration Management Tools empfohlen. Der Kunden hat sich dann für Subversion entschieden, das in diesem Fall nicht im Lieferumfang enthalten ist, da es als Open Source verfügbar ist.

Code-Qualität

Auf Grund der Forderung von Kunden nach der Einhaltung des MISRA Standards, wurde eine Laufzeitumgebung und ein Framework als Basis für die Codegenerierung aus der UML ausgewählt, das MISRA 2004- konform ist.

Parallel dazu wird auch die Qualität des neu erzeugten Codes mit Hilfe des statischen Analysers PC-Lint auf Qualität und MISRA-Konformität überprüft.

Auf Basis der UML lassen sich System-Tests sehr viel einfacher durchführen und automatisieren, da es Testwerkzeuge gibt, die UML-Modelle interpretieren und ansteuern können. Die Einführung von automatischen Tests, wurde jedoch zu Gunsten einer möglichst kurzen Einarbeitungszeit, in eine zukünftige Projektphase verschoben.

Time-to-Market

Die Realisierung dieses Projektes steht unter extremem Zeitdruck. Parallel wird eine große Komplexität erwartet. In der Vergangenheit wurde die Planung immer wieder durch Änderungswünsche in letzter Minute verzögert.

Aus diesem Grund wird in dieser Umgebung auf Rapid Prototyping gesetzt. Im Lieferumfang dieser Umgebung ist ein Evaluation Board von Luminary Micro, inklusive einem Satz an Treiberbibliotheken enthalten. Dazu passend wurde ein Hardware Abstraction Layer (HAL) auf UML Ebene spezifiziert und exemplarisch für einige Komponenten der Hardware-Peripherie implementiert.

Dies ermöglicht es, sehr schnell einen Prototypen zu erstellen, um Interfaces und Bedienkonzept zu testen. Die Entscheidung über die endgültige Hardware-Konfiguration kann in spätere Projektphasen verschoben werden, es muss dann nur noch die unterste Ebene der Treiber angepasst werden.

Ebenso einfach ist die Vorgehensweise, wenn Komponenten einer Applikation in anderen Projekten verwendet werden sollen. Über den HAL sind sie ideal entkoppelt von Hardware und Laufzeitarchitektur. Damit wird das Management von Produktvarianten erleichtert und weitere Produktvarianten können in kürzerer Zeit realisiert werden.

Einarbeitung

Trotz der Komplexität der Entwicklungsumgebung, wurde versucht die Einarbeitungszeit auf wenige Wochen zu begrenzen. Um dieses zu sichern, werden 5 Tage vor Ort Coaching eingeplant.

Ebenso aus Gründen der Einarbeitung wird der IAR Compiler in Kombination mit dem RTOS embOS eingesetzt, da diese Werkzeuge aus vorherigen Projekten bekannt sind. Die vorhandenen Erfahrungen können weiter genutzt werden und die Homogenität der Entwicklungsumgebungen über verschiedene Projekte hinweg bleibt erhalten.

Bestandteile

- Embedded UML-Studio™
- Bridge UML to IDE (Codegen. ARM)
- IAR Compiler IDE
- embOS RTOS (Object Code)
- Luminary Micro Board
- PC-Lint
- ARM-Schulung
- Embedded UML Start-Up Training
- 5 Tage Proj. Start Up Coaching ARM-Schulung

Gesamtkosten ca. 28.000 EUR

Variante 4

Diese Variante basiert auf folgenden Anforderungen eines Kunden:

Hardware Architektur: das geplante Projekt wird in einer nicht sehr großen Stückzahlen (ca. 1.000 pro Jahr) produziert werden. Der Endpreis liegt bei ca. 5.000,- €. Damit fallen die Kosten für die Rechner-Architektur nur schwach ins Gewicht und sind gegenüber anderen Kosten (z.B. Personalkosten) zu vernachlässigen. Es wird Speicher und Rechenleistung in ausreichendem Umfang zur Verfügung gestellt.

Echtzeitanforderungen: es gibt Nebenläufigkeiten mit hohen Echtzeitanforderungen. Der Hauptanteil der Applikation hat jedoch vernachlässigbare Echtzeitanforderungen.

Intellectual Property (IP): in dem bisherigen und zukünftigen Produkt steckt ein großer Anteil an IP. Die Vergangenheit hat gezeigt, dass die direkte Zusammenarbeit mit Kunden und das Design der Funktionalität dicht an deren Bedürfnissen zur Marktführerschaft verholten haben. Die Informationen über viele Lösungsansätze und Funktionen stecken überwiegend in den Köpfen der Entwickler. Das macht es immer wieder schwierig, neue Mitarbeiter in die Entwicklung zu integrieren. Aus dem selben Grund wurde bisher der Schritt einer Neuentwicklung immer wieder gescheut, da schlecht abzuschätzen ist welcher Umfang und welches Risiko mit einer Neuentwicklung verbunden ist. Zukünftig sollen diese Informationen besser dokumentiert werden.

Inzwischen lässt sich die alte, sehr destrukturierte Software nicht mehr effizient warten und ändern. Die Problematik liegt nicht so sehr darin, dass ein komplett neues Produkt benötigt wird, sondern darin, dass die Entwicklungsabteilung die permanente Weiterentwicklung auf Basis der existierenden Software nicht mehr gewährleisten kann und es wurde entschieden parallel zum alten Produkt mit einer Neuentwicklung zu starten.

Lebensdauer des Produktes: sind mind. 10 Jahre. (Das aktuelle Produkt ist seit 17 Jahren am Markt) Erfahrungsgemäß steigen die Anforderungen in dieser Zeit kontinuierlich. Weiterentwicklungen müssen auf Basis von ausgelieferten Produkten geschehen. Wartbarkeit und Erweiterbarkeit muss über den ganzen Lebenszyklus gesichert sein. Es wird mehrere Varianten des Produktes geben.

Time To Market: Es gibt keinen konkreten Zeitplan. Eher ist zu überlegen: wie können Neu-Entwicklung und Pflege des aktuellen Produktes ineinander überfließen. In wie weit macht Reuse alter Sourcen Sinn.

Qualität: Ausfälle des Systems können zu hohen Regressansprüchen führen. In der Vergangenheit hat es immer wieder Qualitätsprobleme gegeben, die aber immer glimpflich verlaufen sind. Dieser Zustand sollte

zukünftig verbessert werden. Ebenso soll in Bezug auf eine mögliche anstehende Zertifizierung gegen die IEC 61508 Vorsorge getroffen werden.

Erfahrung: in der Anwendung der Hochsprache ANSI-C auf verschiedenen Hardwareplattformen ist vorhanden, jedoch keine Erfahrung mit der ARM Architektur. Erfahrungen im Einsatz eines RTOS oder anderen Werkzeugen zum Architekturdesign sind nicht vorhanden. Für Configuration Management ist Telelogic Synergy® im Einsatz, es wurde bereits entschieden Telelogic Doors® als Requirement Engineering Tool einzuführen.

Budget: das Management ist sich dessen bewusst, dass das Software Engineering existentiell für die Firma ist. Es ist bereit für größere Investitionen, wenn IP und Qualität damit gesichert werden können.

Auswahl der Entwicklungsumgebung orientiert an den obigen Anforderungen

Diese Umgebung, die den derzeitigen Stand der Technik abdeckt, umfasst alle Komponenten der Variante 3. Aufgrund der höheren Qualitätsansprüche wird die Umgebung um die Automatisierung von Tests auf Modellebene, so genannte Regression-Tests, erweitert.

In einem weiteren Schritt, kann schon in einer sehr frühen Projektphase, die Simulation der Architektur auf Basis eines Web Interfaces durchgeführt werden. Bereits nach kurzer Zeit kann auf Basis des so genannten MDD-Ansatzes (Model Driven Development), ein ausführbares Modell erstellt werden, auf dem erste Acceptance Tests, z.B. mit Anwendern oder dem Produkt Marketing, durchgeführt werden können. Ebenso können darauf basierend Schulungen des technischen Personals durchgeführt werden, um den Umstieg vom alten System zum Neuen vorzubereiten.

Qualitätssicherung

Qualitätssicherung auf Basis eines Standards wird für mehr und mehr Entwicklungen gefordert, besonders in sicherheitskritischen Bereichen. Diese Umgebung stellt die Mittel zur Verfügung, die entsprechend der EU Norm IEC 61508 für eine Qualitätssicherung gefordert sind: vom Requirement Engineering bis zur Testautomatisierung und der Traceability der Requirements durch das ganze Projekt. Aber auch der handwerklichen Qualität des eigentlichen Codes, wird auf Basis des MISRA 2004 Programmierungsstandards, Beachtung geschenkt.

Sicherung des Intellectual Property (IP)

Zur Sicherung des IP wird Requirement Engineering eingesetzt. Diese Lösung bietet alle Hilfsmittel, um mit einem Req. Engineering Tool zu arbeiten. Parallel dazu werden Tests auf Basis von Requirements automatisiert. Diese Umgebung ermöglicht Requirement-, Software- und Qualitäts- Engineering orientiert am heutigen Stand der Technik.

Allerdings ändert sich durch deren Einführung nicht nur das Handwerkszeug wie Tools, Notationen etc., sondern gleichzeitig müssen sich damit auch die Arbeits- und Denkweisen der Entwickler verändern. Die Phasen im Vorgehen müssen strikt eingehalten werden im Sinne des V-Modells.

Das bedeutet nicht nur die Einarbeitung in neue Techniken, sondern auch die Aneignung neuer Vorgehensweisen durch unterschiedliche Trainingsmaßnahmen, die sinnvoll auf die verschiedenen Projektphasen über mehrere Monate verteilt sind.

Die genannten notwendigen Umstellungen erfordern natürlich einen zeitlichen Mehraufwand. Dieser wird gerechtfertigt durch eine Absicherung des IP, des Qualitäts-Standard und die langfristige Absicherung kurzer Innovationszyklen.

Bestandteile

- Embedded UML-Studio™
- Bridge UML to IDE (*Bethooven*)
- TestConductor
- ValuePack
- Embedded OO-RTX™
- Keil Compiler IDE
- PC-Lint
- Embedded UML Start-Up Training
- 2 Tage Requirement Based Projekt Start-Up Workshop
- 2 Tage UML Proj. Start-Up Coaching
- 2 Tage TestConductor Start Up Workshop
- ARM-Schulung

Gesamtkosten ca. 42.000 EUR

Grundsätzliches

Wir raten grundsätzlich zum Einsatz von folgenden Tools, wenn nicht bereits im Einsatz, auch als Open Source verfügbar sind:

Configuration Management

Wir raten grundsätzlich zum Einsatz eines Configuration Management Tools. Es gibt nahezu nur Vorteile und keine Nachteile.

Hierfür gibt es eine große Auswahl an Werkzeugen. Wenn keine besonderen Anforderungen existieren, ist z.B. der Einsatz von Subversion als Open Source-Variante eine Möglichkeit, die das Budget für Entwicklungstools nicht so sehr belastet.

Wenn auf C-Code Ebene gearbeitet wird

Die Konsistenz zwischen Pflichtenheft (Requirements), Dokumentation, Code und Tests stellt sich immer wieder als eines der Hauptprobleme bei den heutigen Vorgehensweisen heraus.

Aus diesem Grund raten wir grundsätzlich zu Werkzeugen, die helfen alle notwendigen Dokumente die im Software Engineering entstehen, konsistent zu halten. Der Einsatz der UML mit einem guten UML- Tool bietet die beste Voraussetzung. Wird auf ANSI-C-Ebene programmiert, können folgende Tools helfen Dokumentation und Code konsistent zu halten.

- Grafische Editoren, die die Code-Verstehbarkeit erhöhen. Zum Beispiel Easy Code. Auf diesem Weg kann die Dokumentation auch direkt im Source-Code erfolgen und mit so genannten Ausblendungen versteckt gehalten werden. Bei Bedarf kann die Dokumentation eingeblendet werden, so dass der Code an sich nicht auf Grund zu viel integriertem Text an Verstehbarkeit leidet. Auf diese Weise entfällt parallele Dokumentation. Diese Vorgehensweise hat den Nachteil, dass es kein separates Dokument mit der Dokumentation gibt.
- Tools, die helfen Dokumentation und Code konsistent zu halten, indem aus dem Source die Dokumentation automatisch erzeugt wird: Beispiel: Doxygen

WILLERT SOFTWARE TOOLS GMBH ■

Tel. +49 5722 9678 60 - FAX: +49 5722 9678 80
Email: info@willert.de - www.willert.de
Hannoversche Str. 21 - 31675 Bückeburg