

Komplexität managen - modellieren statt programmieren

Newsletter No 22

WILLERT.

Im September 2008 eröffnete Prof. Dr. Annette Schavan (*Bundesministerin für Bildung und Forschung*) die „InnoVisions-Days - Embedded Systems“ unter Anderem mit folgenden Worten:

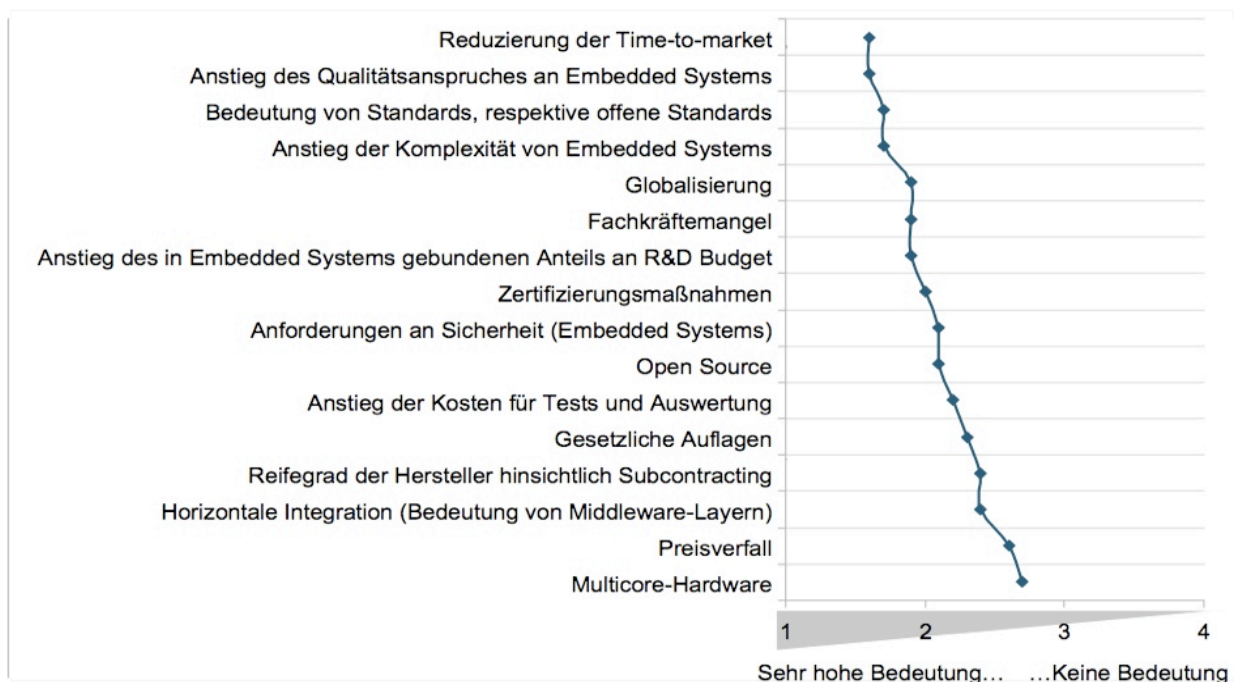
„... Die Informations- und Kommunikationstechnologien (IKT) sind der Innovationsmotor Nummer eins. Mehr als die Hälfte der Industrieproduktion und mehr als 80 Prozent der Exporte Deutschlands hängen heute vom Einsatz moderner IKT ab. Sie sind der Schlüssel für Wachstum und Beschäftigung, für die Wettbewerbsfähigkeit Deutschlands und nicht zuletzt für neue Arbeitsplätze. ... Gerade den Embedded Systems, den elektronischen Zwergen in Gestalt von Minicomputern, die im Verborgenen Funktionsabläufe kontrollieren und steuern, kommt eine Schlüsselfunktion quer durch alle Technologiebereiche zu. Wer hier einen technologischen Vorsprung erzielen kann, hat auf dem Markt einen entscheidenden Wettbewerbsvorteil. ...“

Aber diesen technologischen Vorsprung zu erzielen ist nicht ganz so einfach. Die Anforderungen an die Embedded Systeme und dort vor allem an das Software Engineering sind in den letzten Jahren enorm gestiegen.

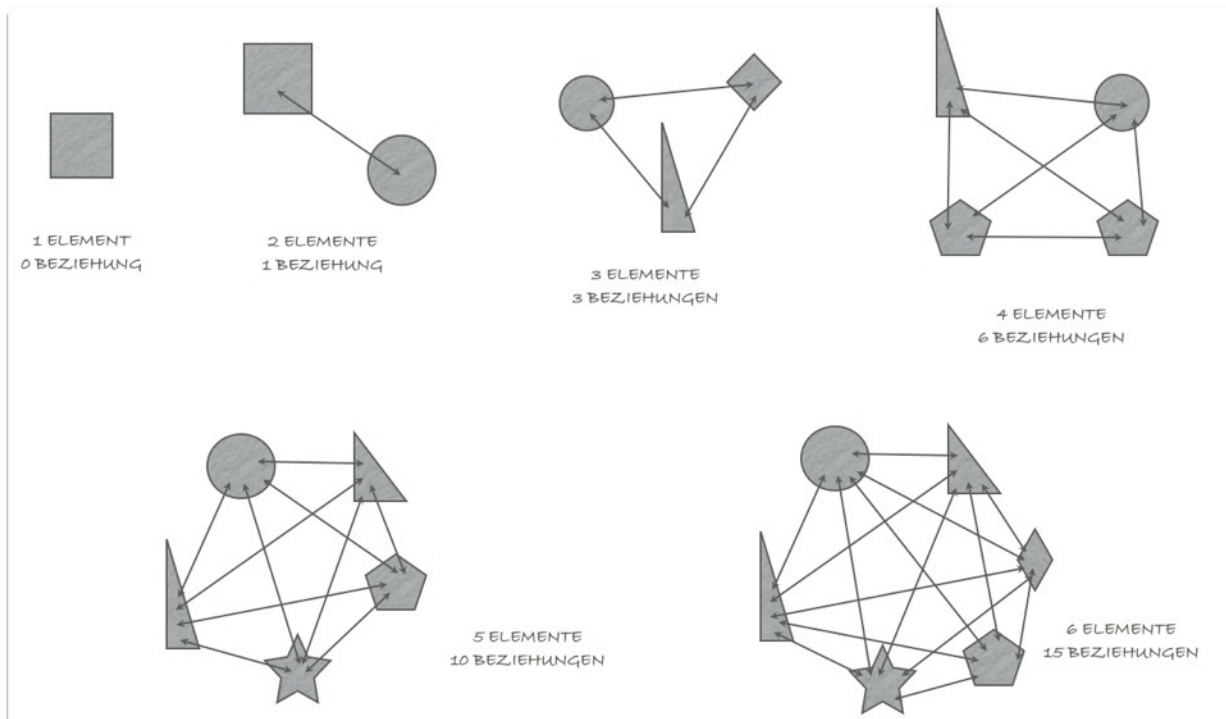
Eine vom Bundesministerium für Bildung und Forschung in Auftrag gegebene Studie bei der BITCOM zeigt, dass für deutsche Unternehmen Time-to-market, Anstieg des Qualitätsanspruchs und Anstieg der Komplexität die bedeutendsten Anforderungen, die es derzeit zu bewältigen gibt, keine leichte Herausforderung sind.

Bereits in der letzten Newsletter haben wir uns mit dem Thema Komplexität beschäftigt und wie steigender Komplexität im Software Architektur-Design effizient begegnet werden kann. In dieser Ausgabe wollen wir weitere Möglichkeiten zeigen. Zum einen im Bereich des Anforderungs - Management (*Requirement Engineering*) und zum anderen in der Modellierung statt Programmierung von Software.

Um mögliche Lösungsansätze besser verstehen zu können möchten wir zuerst noch einmal einen Aspekt der Komplexität beleuchten.



Wachstum der Komplexität. Was ist das?



In obiger Abbildung ist das Verhältnis von Elementen und Beziehungen dargestellt. Die im Verhältnis zu den Elementen exponentiell ansteigenden Beziehungen sind eine oft unterschätzte Ursache für die wachsende Komplexität.

Betrachten wir eine typische Applikation über die Zeit, dann stellt sich heraus, dass im Ursprung oft nur wenige Module existierten. Heute sind diese mit vielfältiger Funktionalität erweitert. Angefangen bei Ferndiagnose, über nachträgliche Software Update Möglichkeiten, grafische Bedienoberflächen, bis hin zu intelligentem Energie-Management steigen die Anforderungen und damit verbunden die Elemente (Operationen und Objekte oder Module) unserer Applikationen.

Die Anzahl der Module stellt dabei das untergeordnete Problem dar. Die Beziehungen der Module repräsentiert die eigentliche Komplexität.

Wenn das obige Bild in Präsentationen gezeigt wird hört man häufig den Einwand, dass ja nicht jedes Modul eine Beziehung zu jedem anderen hat. Bezogen auf eine Ebene mag das richtig sein, aber in heutigen Applikationen haben wir mehrere Ebenen. Im Schnitt etwa 7 (Siehe auch in unserer Newsletter No 21 zu diesem Thema).

Potentiell hat also jedes Modul parallel auf 7 (+/- 3) Ebenen Beziehungen zu jedem anderen und auf jeder Ebene können es wieder mehrere Beziehungen sein. Über alle Ebenen hinweg existiert ein unglaubliches Beziehungsgeflecht. Und darin liegt heute die Hauptanforderung in der Beherrschbarkeit von Projekten.

Ein Beispiel soll das verdeutlichen. In einer Applikation wird ein mechanischer Antrieb abgekündigt. Die neue Version hat modifizierte Antriebsdaten und die Software muss daran angepasst werden. Gleichzeitig soll sie aber auch noch mit den alten Antrieben arbeiten, da in allen Geräten unabhängig vom Stand des Antriebsmoduls die selbe Firmware eingesetzt werden soll.

Eine frühere Anforderung war, dass in einer speziellen Variante das Gerät für Batteriebetrieb mit eingeschränktem Leistungsumfang und einem besonders effizienten Antrieb ausgerüstet wird. Auch diese Variante hat spezielle Anforderungen, die in der selben Firmware unterstützt werden müssen.

Wie kann nun sicher gestellt werden, dass in all den Änderungen gleichzeitig daran gedacht wird, dass diese auch kompatibel zu allen Varianten und Versionen durchgeführt werden?

```

void
getout(r)
{
    int
    r;
    {
        exiting = TRUE;

        /* Position the cursor on the last screen line, below all the text */
#ifdef USE_GUI
        if (!gui.in_use)
#endif
        windgoto((int)Rows - 1, 0);

#ifdef AUTOCMD
        apply_autocmds(EVENT_VIMLEAVEPRE, NULL, NULL, FALSE, curbuf);
#endif

#ifdef VIMINFO
        if (*p_viminfo != NULL)
        {
            /* Write out the registers, history, marks etc, to the viminfo file */
            nsg_didany = FALSE;
            write_viminfo(NULL, FALSE);
            if (nsg_didany) /* make the user read the error message */
            {
#endif
#endif
}
}
-- VISUAL --                               1618,36          98%

```

In obiger Abbildung ist dargestellt, wie in heutigem Sourcecode, zum Beispiel durch bedingte Compilation, einige dieser Ebenen und Beziehungen wiedergespiegelt werden.

All diese Ebenen mit ihren Verflechtungen allein im C-Sourcecode darzustellen ist längst an die Grenzen der Verstehbarkeit geraten.

Aus diesem Grund werden parallel zum C-Code Zusammenhänge von Ebenen evtl. mit Visio gezeichneten grafischen Abbildungen dargestellt. Diese redundanten Darstellungen müssen aber bei Änderungen separat gepflegt werden, was im Alltagsstress meist nicht gemacht wird. In der Folge entspricht die Darstellung dann nicht mehr dem Code.

Ein weiteres Problem ist heute oft die unzureichende Dokumentation der Gesamtheit aller Anforderungen an ein System.

Soll ein System neu programmiert werden ist es meistens unmöglich die inhärent in dem alten System implementierten Anforderungen (*vor allem auch Abhängigkeiten*) vollständig zu erfassen. In der Regel sind sie über die Zeit in das System hinein implementiert worden, ohne jegliche Definition bzw. Dokumentation und nur noch implizit im Sourcecode und in den Köpfen der alt-eingesessenen Entwickler enthalten.

Wahrscheinlich ist diese Situation einer der Hauptgründe, warum eine große Ehrfurcht vor existierenden Systemen besteht verbunden mit der unüberwindlichen Scheu diese neu zu programmieren. Intuitiv ist jedem beteiligten Entwickler klar, dass die Applikation zu einem permanent schwieriger beherrschbaren Beziehungsgeflecht angewachsen ist und der Zustand mit jeder weiteren Änderung bzw. Erweiterung schlimmer wird, aber der Mut für eine grundlegende Neuprogrammierung fehlt. Sicherlich auch, weil vom Management die dafür notwendige Zeit nicht eingeräumt wird. Aber wie soll es enden?

Würden alle Anforderungen bekannt und genau beschrieben sein, wäre eine neue Implementation mit heutigen modernen SW Engineering -

Möglichkeiten gar nicht so aufwändig wie oft befürchtet.

Zwei Engineering-Mechanismen mit den dazugehörigen Werkzeugen helfen diesen Zustand zu verbessern.

- Requirements Management für das Erfassen und Verwalten der Anforderungen und den damit verbundenen Beziehungen.
- Der Umstieg von der Programmierung zur Modellierung von Software, um damit Systeme in wesentlich kürzerer Zeit und besserer Struktur und Dokumentation zu erstellen.

Im Folgenden werden diese beiden Engineering - Mechanismen näher vorgestellt.

Anforderungs-Management

Diese Engineering-Disziplin ist schon lange bekannt. Früher wurde es Lastenheft und Pflichtenheft genannt, und das Standard-Werkzeug für diese Disziplin ist MS Word.

In einfachen Systemen war es möglich alle Anforderungen linear hintereinander weg zu schreiben. Je komplexer das obige Beziehungsgeflecht wird, desto stärker gerät das herkömmliche Pflichtenheft, mit MS Word erstellt, an seine Grenzen.

Rapid Prototyping oder Executable Specification werden herangezogen, aber auch damit lässt sich obige Problematik nicht an den Wurzeln packen.

Die grundlegende Lösung ist ein Anforderungs - Management mit Betonung auf Management.

Damit ist gemeint, dass die Anforderungen nicht nur definiert werden, sondern über den gesamten Software Lifecycle und über alle Phasen des Software Engineering Prozesses verfügbar sind.

Eine dafür notwendige Haupteigenschaft ist das Verlinken von Anforderungen. Zum einen untereinander, um gegenseitige Abhängigkeiten darzustellen, aber auch in die Dokumente und Werkzeuge aller Phasen des SW Engineerings.

Zum Beispiel von der Definition in das UML- Modell hinein an die Stellen, an denen eine Anforderung implementiert ist. Oder in den Testfall hinein, in dem diese Anforderung getestet wird.

Das hilft, die Systeme in sich konsistent zu halten, auch wenn sich die Änderung einer Anforderung an 7 verschiedenen Stellen in der Implementierung und in 3 Testfällen auswirken kann.

Welchen wirklichen Aufwand eine Änderung verursachen wird ist heute in den wenigsten Projekten mit hinreichender Genauigkeit und mit vertretbarem Aufwand analysierbar.

Die Möglichkeit der Verlinkung von Anforderungen in das Design und Testsystem helfen Aufwandsabschätzungen und Änderbarkeit zu verbessern.

Aber auch umgekehrt helfen die Links zu den Requirements. Wenn Sie eine Routine vor sich

haben, in der Sie Änderungen durchführen müssen, sind Ihnen dann alle Anforderungen, die im Lauf der Zeit in diese Routine implementiert wurden, bewusst? In den meisten Fällen existiert wahrscheinlich ein latentes Risiko, dass die Änderung eine inhärente Anforderung nicht berücksichtigt. Ein nun entstandenes Fehlverhalten zeigt sich hoffentlich im Test. Evtl. aber auch erst bei der Inbetriebnahme oder noch schlimmer: im Betrieb des Anwenders.

Je filigraner Anforderungen in die Implementation und den Test verlinkt werden können, desto sicherer können Änderungen, Erweiterungen oder neue Varianten implementiert werden. (Übrigens auch eine Forderung der Norm EN 61508)

Die steigende Anzahl an Beziehungen heutiger Systeme stellt ein außerordentliche Herausforderung an die Konsistenz der Anforderungs-Dokumente dar. Textuell mit MS Word geschriebene Anforderungen (Pflichtenhefte) geraten hier zunehmend an ihre Grenzen.

Die Lösung bieten Werkzeuge. Eines der marktführenden Requirement Engineering - Werkzeuge ist DOORS® von IBM® Rational®.

Mit Hilfe eines Werkzeugs wie DOORS® können Anforderungen nicht nur definiert, sondern in andere Werkzeuge verlinkt werden. Aber auch Änderungen von Requirements können verfolgt werden, Abhängigkeiten dargestellt und Diskussionen zu Änderungen angestoßen, eben die Anforderungen im gesamten Lebenszyklus gemanaged werden. Das ist Requirement Management.

Requirement Traceability

Requirement Management-Werkzeuge wie z.B. DOORS® ermöglichen also das Verlinken von Anforderungen. Das macht jedoch erst richtig Sinn, wenn auch die übrigen Werkzeuge im Software Engineering-Prozess die Links zum Requirement Engineering Tool verstehen und handhaben können.

Hier stoßen wir auf einen Vorteil der UML gegenüber herkömmlichen Notationen wie C, C++

oder Java. Die UML kennt ein Notationselement vom Typ ‚Requirement‘. Es ist bekannt, dass redundante Informationen mehrfach zu pflegen, hohen Aufwand bedeutet bzw. in der Praxis schnell zu inkonsistenten Daten führt.

Ein gutes UML-Modellierungs-Werkzeug basiert aus diesem Grund auf einem Repository. Das ist im Grunde eine Datenbank, in der die Informationen notationspezifisch abgelegt werden. Verschiedene Sichtweisen zeigen im Grunde immer den selben Inhalt des Repositories nur in anderen Zusammenhängen. Es entstehen keine redundante Daten die doppelt gepflegt werden müssen trotz verschiedenen Aspekten der Sichtweisen.

Auf diese Art können auch die Requirements im Repository abgelegt und mit den jeweiligen relevanten Stellen des Modells verlinkt werden.

Ein UML Tool wie IBM® Rational® Rhapsody® beispielsweise kann dann mit DOORS® kommunizieren und die Linkinformationen austauschen. Damit stehen in DOORS® die Informationen zur Verfügung, wo ein Requirement im SW-Design oder in der Implementierung einfließt.

Aber auch in Rhapsody® stehen die Informationen zur Verfügung welches Architektur-Element oder welche Implementation auf welchen Requirements basiert.

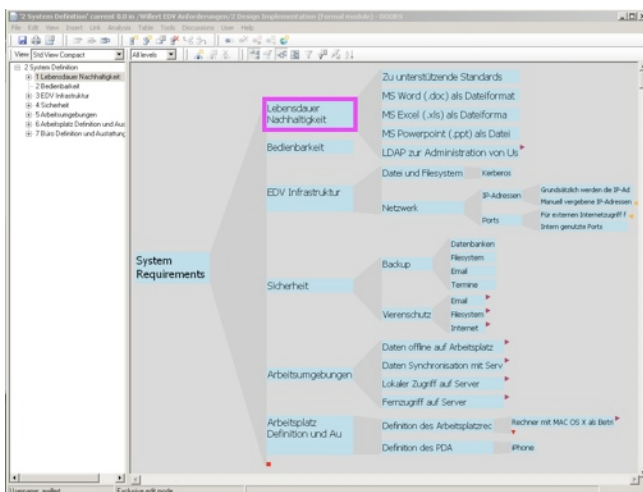
Und es geht noch weiter. Kann das Testwerkzeug auf das Repository zugreifen, dann können auch die Tests mit den Requirements verlinkt werden.

Vielleicht können Sie sich vorstellen, dass echtes Anforderungs-Management nur mit einem durchgängigen System effizient möglich ist, bzw. dass das schreiben eines Pflichtenheftes noch lange nichts mit Anforderungs ‚Management‘ zu tun hat.

Kommen wir noch einmal auf die Komplexität zurück. Wenn wir eine Applikation von den Anforderungen bis zur Implementation betrachten, dann werden wir sehr viele gegenseitige Abhängigkeiten finden. Entsprechend der ersten Abbildung in dieser Newsletter. Jedes Element hat eine Beziehung zu jedem anderen Element. In der Praxis können es sogar mehrere Beziehungen sein.

Eine der Hauptkomponenten des Requirement Managements ist es dieses Beziehungsgeflecht auf Basis von Links nachzubilden. Das verfolgen dieser Links wird als Requirement Traceability bezeichnet. Auf Basis der Traceability innerhalb diesem nachgebildeten Modell kann nun sehr viel sicherer entwickelt werden und unerwünschte Effekte können vermieden werden.

Das hilft dann die Komplexität besser zu managen.



Mit UML von der Programmierung zur Modellierung - eine Stufe in der Evolution des Software Engineering?

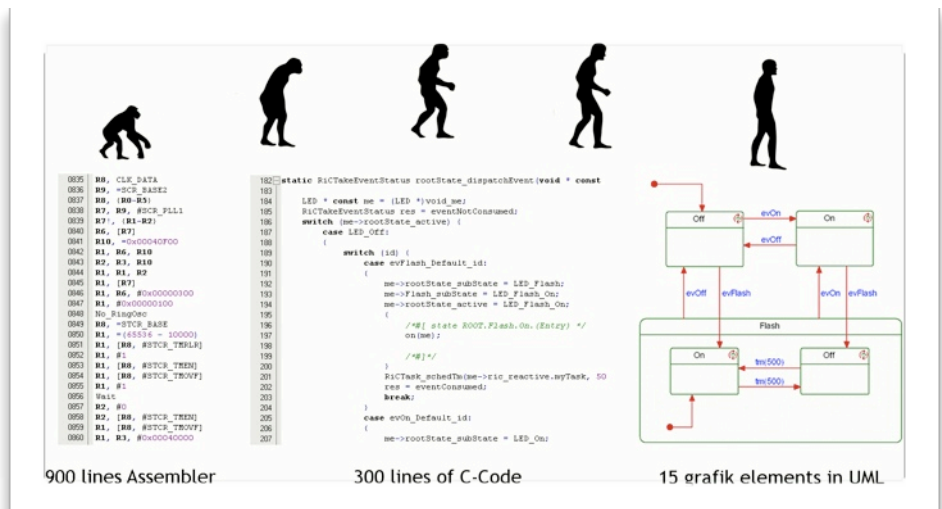
Die Evolution in der Technik schreitet mit riesigen Schritten voran. Wie sieht die Antwort im Software Engineering aus? UML ist die konsequente Weiterentwicklung der Notationen (Programmiersprachen) entsprechend den gestiegenen Anforderungen im Software Engineering und den sich daraus ergebenden Änderungen der Programmier-Techniken. Vergleichbar mit der Entwicklung von Assembler zu Hochsprachen wie ANSI-C.

Wenn wir uns die Entwicklung des Software Engineering und die dazugehörigen Techniken anschauen, dann stellen wir fest, dass Hochsprachen, wie zum Beispiel ANSI-C, die Anwendung einiger dieser erfolgreichen Techniken nicht explizit unterstützt.

Ein Beispiel sind Notationselemente zur einfacheren Anwendung der Objektorientierten Programmierung, die zur Notation C++ geführt haben. Diese sehr offensichtliche Weiterentwicklung ist vielen Entwicklern bewusst. Darüber hinaus gibt es aber viele weitere Techniken, die dem Entwickler häufig nicht so bewusst sind. Dazu ein Beispiel:

Jedes Software-Programm hat implizit eine so genannte Architektur. Zum einen ist das die statische Aufteilung in kleinere Komponenten. Das wird in C auf Basis von Modulen und Unterprogrammen erreicht. Zum anderen ist es die dynamische Ausführung dieser Teilsysteme. Im simpelsten Fall geschieht das innerhalb einer main() Loop, in der die Unterprogramme nacheinander aufgerufen werden. Die modulare Aufteilung eines Systems in Kombination mit einer main() Loop ist eine minimale Software-Architektur. Der Nachteil dieser Architektur ist die schlechte Änderbarkeit hinsichtlich Prioritäten und Zeitverhalten. Um hier effizienter programmieren zu können werden z.B. Echtzeitbetriebssysteme eingesetzt.

Diese werden jedoch nicht explizit von C unterstützt sondern in Form von Bibliotheken als Erweiterung des Befehlsvorrates von ANSI-C geliefert. Der Nachteil: Jeder RTOS-Lieferant hat seine eigene Notation erfunden.



Die UML ist nun der konsequente Schritt Notationselemente zu definieren, die sowohl den Umgang der Notation herkömmlicher Hochsprachen abdeckt, als auch Elemente für darüber hinausgehender praxiserprobter Software Engineering-Techniken.

So ist zum Beispiel eine Klasse in der UML das, was in ANSI-C ein Modul ist. Aber das Notationselement der Klasse geht funktional weit über das Notationselement Modul hinaus. So kann zum Beispiel eine Klasse kooperatives oder preemptives Verhalten haben. Preemptiv bedeutet, dass es innerhalb der Laufzeitarchitektur in einem eigenen Task läuft und andere Klassen unterbrechen kann. Daraus ergibt sich zwangsläufig, dass eine Klasse dann als weitere Eigenschaft auch eine Priorität haben kann.

Kenntnisse über ein spezielles RTOS sind nicht mehr notwendig. Die Bedienung eines RTOS ist implizit in der Notation UML enthalten.

Der daraus entstehende Nutzen ist weit größer, als oft angenommen. Allein bei dem zuvor beschriebenen Beispiel ergeben sich folgende Vorteile:

Dokumentation: Design und Dokumentation verschmelzen zu einer Einheit. Architektur-Elemente, wie in diesem Fall die Design-Entscheidung, ob das Laufzeitverhalten der Klasse kooperativ oder preemptiv ist, sind in der UML Architektur-Darstellung direkt zu sehen (*dokumentiert*). In der ANSI-C - Darstellung stecken sie irgendwo im Sourcecode und sind schwer zu finden.

Verstehbarkeit: Die Modellierung des Laufzeitverhaltens ist in einer standardisierten Form und für jeden, der die Notation UML beherrscht, sofort verständlich dargestellt.

Wiederverwendbarkeit: Die Klasse könnte ohne Änderung in einem anderen Projekt verwendet werden, selbst wenn dort ein anderes RTOS eingesetzt würde, da die Angabe über das Laufzeitverhalten auf Basis der Notation angegeben wurde und nicht auf Basis von nichtstandardisierten Erweiterungen der Notation bzw. in Form von proprietären RTOS - Diensten.

Das ist eine kleine Facette der Elemente in denen die UML weit über ANSI-C hinaus geht. Folgend kurz angerissen einige weitere Beispiele, in denen die grafisch orientierte UML gegenüber herkömmlichen textorientierten Hochsprachen Vorteile hat.

- Durch die grafische Orientierung verschmelzen Programm und Dokumentation. Der daraus resultierende Vorteil liegt auf der Hand, Programm und Dokumentation können nicht mehr inkonsistent werden. *(Dieser Vorteil kann natürlich nur dann genutzt werden, wenn aus dem UML Modell auch automatisch Code generiert wird. Im anderen Fall wäre es, als würde man ANSI-C zur Beschreibung seiner Software nutzen, aber dann die Umsetzung in Assembler zu programmieren.)*
- Die Funktionalität von Echtzeitbetriebssystemen ist implizit in der Notation UML enthalten. Es gibt keine Trennung mehr zwischen ANSI-C - Befehlen und Betriebssystem- Befehlen.
- Die UML beinhaltet alle notwendigen Notationselemente für die Anwendung der Objekt Orientierten Programmierung.
- Die UML besitzt Notationselemente, um die sehr mächtige Methode der grafischen Programmierung auf Basis von FSM (*Finite State Mashines*) anzuwenden.
- Die UML besitzt Notationselemente zur Definition von Requirements. Das ist eine wichtige Voraussetzung für das Verlinken von Requirements mit Design und Implementation. Dieses wiederum erleichtert die sogenannte Traceability von Requirements, eine Forderung z.B. der EN 61508 zur Entwicklung von sicherheitsrelevanter Software.
- Die UML hat Notationselemente, auf dessen Basis sich dynamische Abläufe der Software beschreiben lassen. Damit lassen sich auch Testsequenzen beschreiben und automatisch auf das Modell anwenden. Die Basis zur einfachen Durchführung von automatischen (*Regressions*) Tests.

So weit ein kleiner Auszug aus den Notationselementen der UML. Dieser Auszug gibt Ihnen hoffentlich schon eine ungefähre Vorstellung über das enorme Potential, das in der Anwendung der UML liegt.

Leider wird die UML zur Zeit sehr häufig lediglich zur Dokumentation von Software eingesetzt. Die Bewertung des Nutzen auf diese Weise ist fragwürdig und fällt fast immer negativ aus. Kein Wunder, denn die Investition in die Einarbeitung in die Notation und ein evtl. genutztes Tool steht ein zweifelhafter Nutzen gegenüber, denn es wurde

lediglich die Qualität der Darstellung der Dokumentation verbessert. Ist dieses das Hauptproblem in Bezug auf Software Dokumentation? Nein! Das Hauptproblem liegt darin, dass Dokumentation und Source nicht konsistent gehalten werden.

Erst wenn die UML konsequent als Notation zur Modellierung und Implementation eingesetzt wird (*Vorraussetzung dafür ist natürlich eine durchgängige Werkzeugkette*) ergeben sich daraus die wesentlichen Vorteile.

Embedded UML Studio II

Endlich ist es soweit, das neue Embedded UML Studio™ second Generation ist released.

Basierend auf einer neuen OEM-Version von Rhapsody® konnten wir den Preis auf 4.950,- € senken.

Trotzdem ist alles enthalten, was für die Anwendung der UML in kleinen und mittelgroßen Embedded Projekten notwendig ist.

- Unterstützung aller UML 2.0 Diagramme
- Codegenerierung bereits angepasst für eine große Anzahl Zielplattformen
- Schnittstelle zu anderen Werkzeugen
- Erweiterbar mit dem ‚Embedded UML Target Debugger™‘ zukünftig auch für Regression Test auf Basis des TestConductors
- Inclusive ‚Realtime Execution Framework™‘

Mit diesem Preis besitzt das ‚Embedded UML Studio™‘ ein hervorragendes Leistungsverhältnis und eignet sich ideal für den Einsatz der UML in Embedded Projekten.

Weitere Informationen: www.willert.de/uml

Willert Software Tools besitzt IBM® - Zertifizierungen für die Produkte DOORS® und Rhapsody®

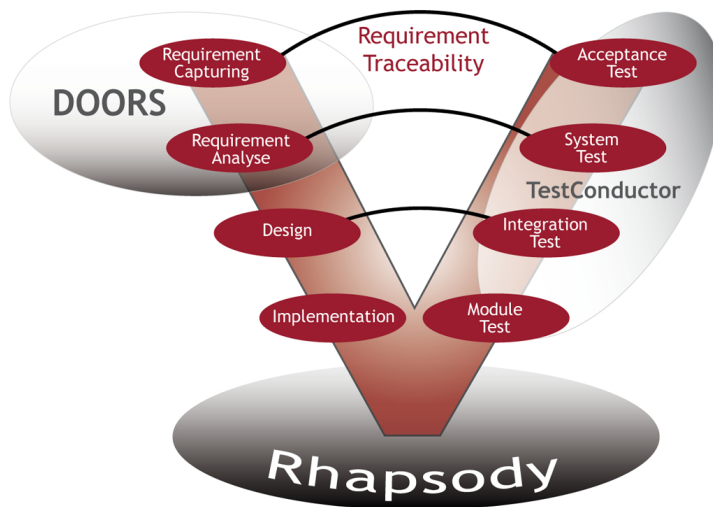
Im September hat die Firma Willert Software Tools technische Zertifizierungen für die IBM® Rational® Produkte DOORS® und Rhapsody® abgeschlossen.

Speziell zum Produkt DOORS® wurde ein besonderes Know How aufgebaut. Damit ist nun auch der Bereich „Requirement Engineering“ mit in das Leistungsspektrum der Firma Willert integriert.

Wenn Sie sich noch intensiver informieren möchten besuchen Sie einen der Workshops zu diesem Thema in Ihrer Nähe.

IBM® Rational® Rhapsody®, DOORS®
IBM® Rational® Test Produkte

Alles was sie benötigen für durchgängiges Software Engineering.
Informationen zu den Workshops unter www.willert.de/termine



Termine und Orte:
2009-10-13 Hamburg
2009-10-28 Düsseldorf
2009-10-30 München
2009-11-12 Osnabrück

Unsere Aktion: Tools + Schulung = Erfolg

Um Ihnen die Entscheidung, in Requirement-Management und/oder Modellierung zu investieren, leichter zu machen und einen erfolgreichen Einstieg zu ermöglichen, gibt es bei Willert bis zum Ende des Jahres 2009 zu unseren Produkten und Schulungen besonders günstige Komplettpakete:

**Embedded UML Studio II
+ UML Start-Up Training**
Authorized User License + SW Subscription,
12 Monate Support
+ 3 Tage UML Start-Up Training in Bückebug
5.950,- € zzgl. MwSt

**IBM® Rational® DOORS®
+ Start-Up Training**
Authorized User License + SW Subscription,
12 Monate Support
+ 2 Tage Doors® Start-Up Schulung in
Bückebug
4.650,- € zzgl. MwSt



UMLforum.de

Herausgeber:

WILLERT SOFTWARE TOOLS GMBH
Hannoversche Straße 21
31675 Bückeburg

www.willert.de info@willert.de

Tel.: +49 5722 9678 - 60