TREND GUIDE

Embedded Software Redesign

MICROCONSULT

TRAINING. COACHING. ENGINEERING.

Durchblick durch Teamfähigkeit:

Software Redesign mit ARTiSAN Studio und UML 2

www.artisansw.com

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Inhalt

Embedded Software Redesign

Vorwort 3

Teil I
Die Entscheidung 5

Teil II
Reverse Engineering: Vom Lösen gordischer Knoten 13

Teil III
Refactoring: Aus Alt mach Neu 19

Teil IV
Reengineering: Der Weg ist das Ziel 26

Info-Pool: Buch-Tipps 33

Info-Pool: Tools und Webtipps 34

MicroConsult Leistungen:
Training, Coaching, Engineering 35

Die Autoren 40

Impressum 41

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Vorwort

Liebe Leser,

schön, dass Sie sich die Zeit für unseren Trend Guide nehmen. Es wird sich für Sie lohnen. Wir haben wieder viele interessante Informationen und Tipps in einen unterhaltsamen Text verpackt, in dem Sie sicher an der einen oder anderen Stelle Ihre eigene Situation wiedererkennen werden.

Ein herzliches Dankeschön an meine Kollegin Sabine Pagler und meine Kollegen Thomas Batt und Frank Listing, die eine Menge Engagement, Sachverstand und Ideen eingebracht haben. Nicht zu vergessen die Gesellschafter von MicroConsult, die uns den Freiraum gaben, diesen Trend Guide zu schreiben.

Martina Hafner und Hans Wiesböck von der Elektronikpraxis danke ich für die Anregungen aus unserem gemeinsamen Projekt "Embedded Software Engineering Report".

Mein besonderer Dank gilt Christiane Kapteina von ARTiSAN Software Tools, die sich dafür eingesetzt hat, dass ihr Unternehmen diesen Trend Guide als Sponsor unterstützt.

Und schließlich danke ich Ihnen, den Lesern, die uns durch viele Downloads und positive Rückmeldungen ermutigt haben, wieder in die Tasten zu hauen. Wir freuen uns auch diesmal wieder über Ihre Meinungen und Anregungen unter trendguide@microconsult.de.

Viel Spaß beim Lesen wünscht

Peter Siwon

p.siwon@microconsult.com

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Liebe Leser,

Software lebt, und nach dem „Prinzip der defensiven Programmierung" wird möglichst lange nur minimal-invasiv in diesen lebenden Organismus eingegriffen.

Das Ergebnis ist häufig eine kaum noch erkennbare Architektur und Struktur des Softwaresystems. Gleichzeitig steigt die Komplexität der Anforderungen, und die Entwicklungszyklen werden immer kürzer. Irgendwann sprengt die nächste gewünschte Änderung oder Erweiterung den Rahmen, ein Redesign muss her!

Hätte man jetzt doch bloß noch irgendwo einen Bauplan oder ein Modell! Dann hätte man wenigstens eine Vorstellung von der ursprünglichen Struktur, ohne sich durch Millionen Lines of Code quälen zu müssen.

Neue Systeme werden heute vielfach bereits objektorientiert konzipiert, modellbasiert entwickelt und auch gewartet, so dass man jederzeit den aktuellen Konstruktionsplan verfügbar hat und Änderungen gezielt an wenigen definierten Stellen einbauen kann.

Aber auch bei der Verjüngungskur für alte Systeme kann die Modellierung helfen. Lesen Sie im MicroConsult Trend Guide alles über Anti-Aging für Ihre Software!

Viel Spaß beim Lesen und anschließenden Modellieren wünscht

Christiane Kapteina

Christiane.Kapteina@artisansw.com

Die 3 Rs des Embedded Software Redesign

Teil I: Die Entscheidung

Auf unseren letztjährigen Veranstaltungen zum Thema **Software Redesign für Embedded Entwickler** wurde aus den Diskussionen und Beiträgen der Teilnehmer sehr schnell klar, wie dieses Thema vielen Firmen auf den Nägeln brennt.

Software wuchs zum Teil über Jahrzehnte unter den Randbedingungen Echtzeit, Speicheroptimierung, Kostenoptimierung und Projektdruck zu immer komplexeren Gebilden. Jetzt stoßen viele Entwicklungsteams an die Grenzen vertrauter Methoden und Vorgehensweisen. Ohne Übertreibung kann man sagen, dass sie nach einem Befreiungsschlag suchen, um wieder Ordnung ins Chaos zu bringen.

Mit diesem Trend Guide wollen wir Sie dabei unterstützen.

Abbildung 1: MicroJones poliert M2C2 nach erfolgreichem Redesign

Guthaben, Kredite und Zinsen auf Software-Qualitätskonten

Das Software-Dilemma lässt sich sehr anschaulich mit einer Analogie beschreiben. Jedes Entwicklungsteam verfügt über drei Qualitätskonten:

- das Dokumentationskonto
- das Architektur-/Source-Code-Konto
- das Prozesskonto

Ist alles nach Lehrbuch gelaufen, sind diese Konten gut gefüllt. Die Dokumentation erlaubt eine schnelle Orientierung in Architektur und Source-Code, und neue Mitarbeiter können sich schnell einarbeiten.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Die Architektur und der Source-Code der Software lassen sich einfach und schnell erweitern. Der Prozess sichert den reibungslosen Ablauf von den Anforderungen bis zur Inbetriebnahme beim Kunden.

Typischer Zustand der Software-Qualitätskonten:

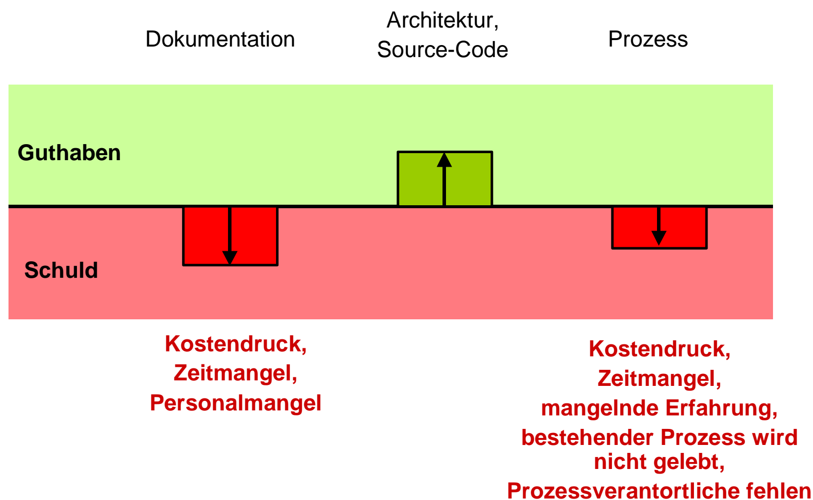

Abbildung 2: Software-Qualitätskonten

Märchenhaft! Genau das ist es leider auch. Ein Märchen, das gerne erzählt und selten gelebt wird. Die Realität sieht meist anders aus und erinnert eher an einen Action Thriller: Einige wenige begnadete „Freaks“ bauen einen Prototypen. Wenn es dafür Kunden gibt, kann der Softwareentwickler ja immer noch alles „sauber“ programmieren. Doch der Kunde will erstens mehr, zweitens anders, und zudem alles schneller. So wird der Prototyp flugs zum Produkt „aufgepeppt“, und es werden noch schnell einige Zusatzfunktionen eingebaut. Das Team ist eingespielt und die Dokumentation überwiegend in den Köpfen gespeichert. Die Architektur ist zwar nicht optimal, bietet aber noch etwas Raum für mehr Funktionalität. Der Prozess gleicht dem virtuosem Jonglieren mit Dateien und Tools.

Von Anfang an sind die Qualitätskonten nur spärlich gefüllt. Wachsende Komplexität, der Verlust von Wissensträgern oder eine Vergrößerung des Teams führen schnell dazu, dass Kredite aufgenommen werden müssen, um dem Projektdruck standhalten zu können. Trotz aller guter Vorsätze wird die Dokumentation nicht nachgeführt, obwohl die Funktionalität stark angewachsen ist. Die Architektur lässt sich nur noch durch zweifelhafte Tricks erweitern und gleicht immer mehr einem Kartenhaus, das beim kleinsten Luftzug einzustürzen droht. Der Prozess besteht zum Großteil aus hektischem „Löcherstopfen“ anstelle systematischer Vorgehensweisen.

Und wie das mit Krediten nun mal so ist: Zinsen fallen an. Nach kurzer Zeit kann keiner mehr die Software überschauen. Immer mehr Klimmzüge und Feuerwehreinsätze sind vonnöten, um die Software zu stabilisieren. Die alten Hasen sind völlig überlastet und die Neuen tappen eifrig, aber wenig zielgerichtet im Dunkeln. Das Management treibt die Entwickler mit der Outsourcing-Peitsche an.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

In unserem Szenario heißen die Zinsen Zeitverlust, Qualitätsverlust, Kundenverlust, Geldverlust, keine Lust — Frust !

Abbildung 3: Gefährliche Kredite

Wie sieht Ihr Kontostand aus?

Unserer Erfahrung nach sind die Qualitätskonten meist viel schneller geplündert, als man es wahrhaben will. Lassen Sie es nicht so weit kommen. Am besten ist es, erst gar keine Schulden zu machen. Folgende Maßnahmen eignen sich zur „Schuldentilgung“:

- **Reverse Engineering** verbessert die Dokumentation
- **Refactoring** sorgt für eine ausbaufähige Softwarearchitektur und erweiterbaren Source-Code
- **Reengineering** führt zu den erforderlichen Prozessanpassungen

Kurz: **die 3 Rs des Software Redesigns.**

Intensivstation oder Pathologie?

Bevor Sie sich nun mit Hilfe der 3 Rs wieder auf den Pfad der Tugend und Schuldenfreiheit begeben, sollten Sie sich folgende Frage stellen: Lohnt sich der ganze Aufwand? Auch wenn es makaber klingt, Sie müssen herausfinden, ob Sie

- die Software am Leben erhalten können,
- nur die brauchbaren Teile weiterverwenden wollen,
- eine Softwareleiche sezieren, um etwas für die Zukunft zu lernen, oder
- gleich alles Begraben und einen Neuanfang wagen.

Zumindest einige im Team haben sicherlich schon das bange Gefühl, dass es so nicht weiter gehen kann. Doch sollte sich eine Entscheidung auf Fakten stützen, die die Situation greifbar machen. So werden auch weniger empfindliche Gemüter ins Boot geholt — vor allem das Management, das die Sache durch die eher distanzierte betriebswirtschaftliche Brille betrachtet.

Eines muss von Anfang an klar sein: Das Füllen der Qualitätskonten ist eine Investition, die zunächst auf Kosten anderer Konten erfolgt — wie zum Beispiel Produktivität oder Gewinn. Die Zinsen, die wir dann allerdings auf unsere Qualitätskonten erhalten, werden diese Investition schon bald wieder in Form kürzerer Projektlaufzeiten und -kosten amortisieren.

Die Diagnose

Willkommen in der medizinischen Abteilung des Software Redesigns! Im Wesentlichen gibt es drei Kriterien, anhand derer wir den Zustand des Softwarepatienten einschätzen können:

- die äußere Qualität der Software
- die innere Qualität der Software
- projektspezifische Anforderungen

Die **äußere Qualität** wird anhand der von außen erkennbaren Eigenschaften bewertet, beispielsweise mit diesen Fragen:

- Welche Betriebszustände oder Veränderungen führen zu Fehlerhäufungen?
- Wie häufig kommen Fehlermeldungen beim Systemtest vor?
- Wie häufig sind Laufzeitfehler nach der Auslieferung?
- Wie umfangreich ist die Funktionalität?
- Wie haben sich die o.g. Zahlen im Vergleich zu den letzten Softwareständen verändert?
- Ist ein Zusammenhang zwischen der größeren Funktionalität und dem Aufwand bis zum problemlosen Einsatz beim Kunden erkennbar?
- Sind alle Anforderungen an Sicherheit, Zuverlässigkeit und Performance erfüllt?
- Ist der Kunde mit der Bedienbarkeit zufrieden?

So könnte sich herausstellen, dass die Einführung neuer Funktionen mit einer sprunghaften Zunahme von Fehlern auch außerhalb dieser Funktion einhergeht, die nur schwer in den Griff zu bekommen sind. Problematische Abhängigkeiten innerhalb der Software könnten die Ursache sein.

Eine weitere interessante Aussage liefert die Dauer der verschiedenen Projektphasen. Wenn sich Systemtests nicht mehr angemessen durchführen lassen, die Kinderkrankheiten neuer Produkte rapide zunehmen oder die Anforderungen ständig über den Haufen geworfen werden, dann deutet dies auch auf Schwächen im Prozess hin. Soweit vorhanden, können hier Fehler- und Ausfallstatistiken aus dem Feld oder auch interne Fehlerlisten aus der Entwicklung wichtige Hinweise geben.

Hier geht es nicht um mögliche Fehler in der Entwicklung, sondern ob die Entwicklungs-Randbedingungen Fehler stärker begünstigen und deshalb Abhilfe geschaffen werden muss.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Geht das System häufig in Überlast, deutet das auf einen Speicher-Engpass oder auf eine zu schwache CPU hin. Speicher- und CPU-Auslastung lassen sich mit den heute üblichen Tools (Linker, Debugger, Emulatoren) in der Embedded Softwareentwicklung bestimmen.

Die **innere Qualität** zielt auf eine Bewertung der Software selbst. Sinnvolle Kriterien:

- Anzahl der Kommentarzeilen
- verfügbare Dokumentation
- Architekturmerkmale: Portierbarkeit, Wartbarkeit, Erweiterbarkeit, Wiederverwendbarkeit
- Komplexität

Neue Mitarbeiter oder Experten außerhalb des Entwicklungsteams liefern nicht selten wertvolle Aussagen. Ein gutes Indiz ist beispielsweise die Zeit, die ein neuer Mitarbeiter für eine Änderung benötigt, oder der „Nervfaktor“: Wie oft muss der neue den erfahrenen Mitarbeitern Fragen stellen, weil ihm die Dokumentation allein nicht mehr weiterhilft? Wenn Änderungen im Vergleich zu früher immer länger dauern und neue Mitarbeiter dafür immer mehr Unterstützung benötigen, ist dies ein sicheres Anzeichen für Handlungsbedarf.

Hilfsmittel für eine grobe Einschätzung sind Tools wie CNCC oder LocMetrics, die das Verhältnis von Kommentarzeilen zu Codezeilen ermitteln (s. Info Pool: Tools und Web-Tipps). Dabei ist jedoch Vorsicht geboten, denn auch ein auskommentierter Code oder nutzloser Kommentar bzw. rein optische Gliederungshilfen, wie Zeilen mit lauter Sternen, werden als Kommentarzeile gewertet.

Ein zumindest stichprobenartiges Überprüfen der Listings ist deshalb unvermeidlich. Oft sind die einzigen Kommentare nur auskommentierte Codezeilen von vorangegangenen Versuchen. Auch finden sich Zeilen wie „It´s magic“ (frei übersetzt: „Ich habe selber keine Ahnung, warum das so funktioniert“) oder Kommentare, die einen Programmierneuling wohl daran erinnern sollen, was „IF“ oder „FOR“ bedeutet. Auch der Witz des Tages oder Statements eines frustrierten Entwicklers („Wer hat denn diesen Sch... geschrieben?“) werden hier und da gefunden. Das mag lustig sein, aber sicher nicht hilfreich. Die Anzahl der Kommentarzeilen ist bekanntermaßen nicht gleichbedeutend mit nützlicher und verständlicher Dokumentation.

Der **innere Aufbau** lässt sich ebenfalls durch ein stichprobenartiges Review einschätzen. Gibt es Monster-Funktionen oder Monster-Codezeilen, bei deren Analyse man Kopfschmerzen bekommt? Wie sieht es mit Einrückungen und anderen optischen Gliederungshilfen aus? Wie stark sind die Programmteile ineinander verflochten? Ein gut lesbarer Code besitzt eine gewisse Ästhetik und Transparenz, wie die „alte Hasen“ predigen.

Einen Maßstab für die Komplexität liefert die zyklomatische Zahl. Der Name alleine flößt schon Ehrfurcht ein, doch was steckt dahinter? Ein gewisser McCabe hat sich dazu den Kopf zerbrochen und folgenden Zusammenhang in eine Formel gegossen:

$$V(G) = e - n + 2p$$

e: Anzahl der Kanten
n: Anzahl der Knoten des Kontrollflussgraphen
p: Anzahl der verbundenen Komponenten

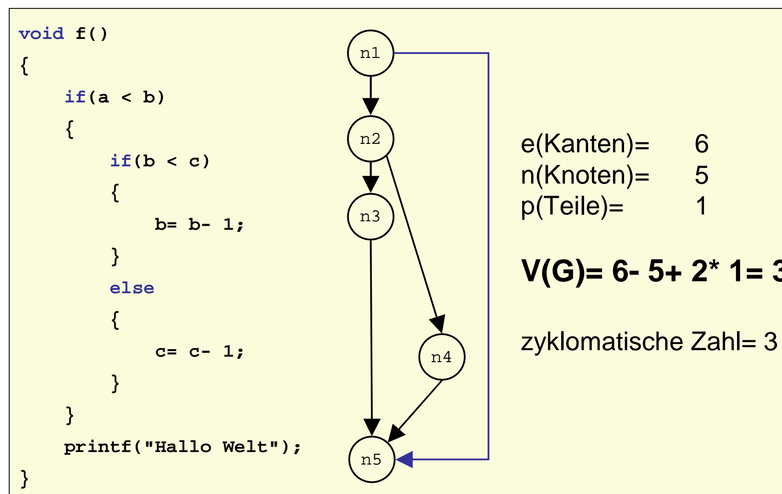

Abbildung 4: Erklärung der zyklomatischen Zahl

Allerdings ist auch diese Formel mit Vorsicht zu genießen. Sie bietet einen großen Interpretationsspielraum, weil eine hohe Komplexitätszahl auf der Ebene des Programmpaketes nicht zwangsläufig unleserlichen oder schlecht strukturierten Code bedeutet. Es kann durchaus die Verstehbarkeit verbessern, wenn die Anzahl der Module oder Unterprogramme erhöht wird. Auf Funktionsebene allerdings ist eine hohe zyklomatische Zahl in jedem Fall ein Grund, den Code genauer unter die Lupe zu nehmen. Ein Tool, das die zyklomatische Zahl abbildet, ist CCCC-C oder CCCC-C++ (siehe Info Pool: Tools und Web-Tipps).

Auch die **Anforderungen aus den Projekten** liefern wichtige Entscheidungshilfen. Wenn ohnehin Veränderungen anstehen – die Wahl eines neuen Prozessors, die Erweiterung des Teams z.B. durch Offshore-Anbieter oder eine starke Veränderung des Funktionsumfanges – sollte man die Gelegenheit gleich nutzen und reinen Tisch machen. Das Thema Outsourcing ist oftmals prekär, da das Zeit- oder Kostenproblem, welches das Management möglicherweise zu diesem Schritt bewogen hat, nicht selten durch hohe Defizite in Dokumentation, Architektur oder Prozess entstand. Ein sinnvolles Outsourcing erfordert zwingend eine Nachführung der Dokumentation und eine Reformierung des Prozesses. Damit aber wäre die Ursache der Zeit- und Kostenprobleme in vielen Fällen weitgehend behoben.

Die einfachste Entscheidungshilfe zur Veränderung sind gesetzliche Vorgaben, Kundenanforderungen oder auch ein innovativer Mitbewerber, der sich bereits entschlossen hat, seine Softwareentwicklung zu reformieren.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Maturity Levels nach CMMI oder SPICE oder Forderungen nach Standards – UML (Unified Modeling Language), MISRA (The Motor Industry Software Reliability Association) etc. – sind hier ebenfalls als Auslöser zu nennen.

Wo immer Sie Tools einsetzen, werden Sie um eine Betrachtung des Codes und der Dokumentation nicht herumkommen, wenn Sie die Aussagen der Tools richtig bewerten wollen.

Der Aufwand für Reverse Engineering und Refactoring lässt sich qualitativ wie folgt einschätzen:

Je niedriger das Abstraktionslevel ist, desto wahrscheinlicher sind die Verflechtungen von Code, Daten und Hardwareplattform. Dadurch erhöht sich naturgemäß der Aufwand für das Reverse Engineering. Ein guter Grund, über objektorientierte und/oder modellbasierende Lösungswege für die nächsten Softwaregenerationen nachzudenken.

Wollen Sie auch quantitative Aussagen über den zeitlichen Aufwand treffen, so sind die folgenden Arbeitsschritte für Analyse und Dokumentation zu bewerten. Dies kann im Team mit Hilfe eines erfahrenen Coaches in 1-2 Tagen erfolgen. Hierzu werden repräsentative Abschnitte ausgesucht, exemplarisch bewertet und der Aufwand auf die gesamte Software hochgerechnet. Die Abschätzung bezieht sich auf Arbeitsschritte wie Vorbereitung, Nachbereitung, Analyse der Architektur, Analyse der Datei- und Funktionsköpfe, Codeanalyse, etc. Der Aufwand für die Codedokumentation lässt sich z.B. mit Erfahrungswerten aus der Fagan-Inspektion abschätzen. Bei schlecht dokumentiertem Code ist eine Architekturanalyse ohne Codeanalyse schwer umsetzbar.

Deshalb hängt der Aufwand natürlich stark davon ab, wie tief und häufig Sie in den Code einsteigen müssen, um die inneren Zusammenhänge verstehen zu können. Die Abschätzung kann im Laufe des Reverse Engineering immer mehr verfeinert werden.

Eine weitere wichtige Entscheidungsgrundlage ist die realistische Einschätzung der verfügbaren Ressourcen für das Redesign. Neben der i.d.R. unrealistischen 100%-Lösung gibt es viele brauchbare Kompromisse für Neustrukturierung, Neucodierung und Prozessanpassung. Sie können sich beispielsweise zunächst nur auf die Teile konzentrieren, die ein besonders hohes Risiko bergen oder die ohnehin angepackt werden müssen. Auch entscheiden Sie selbst, wie feingranular Ihre Maßnahmen tatsächlich angelegt sein sollen, um wieder genügend Licht ins Dunkel zu bringen.

Diese Entscheidungen fordern Ihren gesunden Menschenverstand und fachlichen Instinkt. Tools liefern Hilfen , doch entscheiden muss das Team und unterstützen das Management. Sehr hilfreich können in diesem Prozess erfahrene Projektcoaches sein: Sie bringen Erfahrungen aus Software-Redesigns mit zum Teil ganz unterschiedlichen Randbedingungen mit. Auch nehmen Coaches die Position eines (weitgehend) unabhängigen und sachlichen Beobachters und Moderators ein. Sie sind nicht „betriebsblind“ und zudem keinen starken emotionalen Einflüssen ausgesetzt, da das Projekt nicht ihr „Baby“ ist.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Fazit für sinnvolles Vorgehen beim Redesign:

1. Suchen Sie objektive Gründe für ein Software Redesign, die eine betriebswirtschaftliche Abschätzung von Aufwand und Nutzen ermöglichen.
2. Betrachten Sie die äußere und innere Qualität sowie die Umstände Ihres Softwareprojekts und sammeln Sie dann Pro- und Contra-Argumente.
3. Nutzen Sie Tools, doch setzen Sie Ihr eigenes Know-how dabei sinnvoll ein.
4. Suchen Sie nicht die perfekte, sondern die machbare Lösung.
5. Nutzen Sie Projektcoaches.
6. Letztlich geben immer der gesunde Menschenverstand und der fachliche Instinkt den Ausschlag.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Die 3 Rs des Embedded Software Redesigns

Teil II
Reverse Engineering: Vom Lösen gordischer Knoten

Nach der Analyse von Architektur, Source-Code und Dokumentation und einer Aufwandsabschätzung sind die Teammitglieder gemeinsam mit dem Management zu folgenden Schluss gekommen:

Ein besser dokumentiertes Wissen über die Software ist dringend erforderlich, um ihren Wert zu erhalten und auf dieser Substanz weiter aufbauen zu können. Eine Neuentwicklung ist teurer als die gezielte Verwertung des vorhandenen Codes.

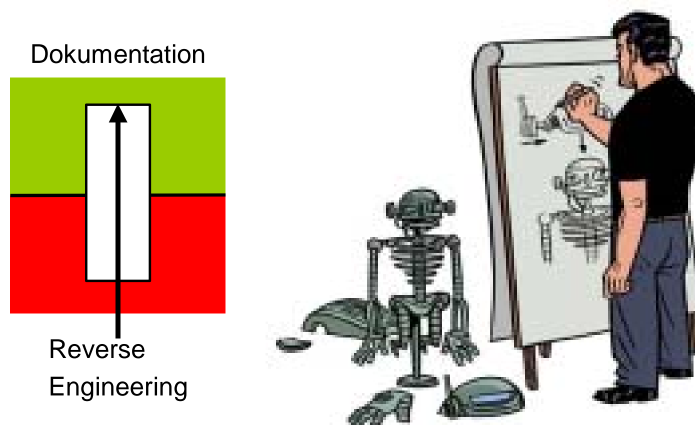

Abbildung 5: Reverse Engineering füllt das Dokumentationskonto

Die jetzt angewandte Methode wird als **Reverse Engineering** bezeichnet. Im Extremfall bedeutet es, dass Sie nur den Code haben und daraus Informationen über Architektur, Konzept, Prinzipien etc. gewinnen und dokumentieren. Das ist vergleichbar mit der Aufgabe, ein stark verknotetes Seil zu entwirren, ohne den gordischen Knoten nach Manier Alexander des Großen einfach mit dem Schwert durchzuhacken.

Die Herausforderung dabei ist, dass Sie erheblich mehr Informationen haben wollen als tatsächlich im Code zu finden sind. Woher nehmen und nicht stehlen?

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Folgendes wissen Sie beispielsweise von Ihrem Code:

- Es gibt zuwenig Informationen über die inneren Zusammenhänge (Architektur und Struktur).
- Es gibt Bereiche, die sich keiner mehr ohne ein Grummeln in der Magengrube anfassen traut.
- Es ist nicht klar, welche Anforderungen umgesetzt wurden und welche nicht.
- Es gibt so etwas wie „Zombiecode": toter Code, der weiterhin ein Schattendasein im Speicher fristet.
- Es ist nicht klar, was zur Weiterverwendung taugt und was am besten neu programmiert werden sollte.

Reverse Engineering hilft Ihnen dabei, diesen Alptraum erst fassbar zu machen und dann loszuwerden. Frei nach dem Motto: Der schnellste Weg zur Software-Besserung ist Software-Kenntnis.

Sie haben es wahrscheinlich schon befürchtet – das geht nicht automatisch. Diese Erkenntnis haben wir schon in Teil 1 gewonnen. Leider (oder zum Glück) gibt es noch keine Tools, die intelligent genug sind, aus dem Code die Motive des Codierers herausdestillieren können. Aus der Formel $E=mc^2$ lassen sich schließlich auch nicht die Gedankengänge Einsteins zur Relativitätstheorie ableiten. Zum Glück ist es bei Software meist nicht ganz so kompliziert. Was Sie vor allem suchen ist:

Übergeordnete Information

- Architektur
- Implementierte Features

Codedokumentation

- Dateiköpfe
- Funktionsköpfe
- Beschreibung von Codeblöcken und Codezeilen

Die Vorgehensweise besteht im Prinzip in drei Schritten:

- Analysieren der Software
- Entscheiden, was mit den identifizierten Softwareteilen passiert
- Dokumentieren der Ergebnisse

Wir unterscheiden zwei Formen der Analyse. Zum einen gibt es die **Architekturanalyse**. Sie kümmert sich um die strukturellen Zusammenhänge des Softwarepaketes auf Modulebene. Die **Codeanalyse** befasst sich mit formellen Dingen wie Dateiköpfen, Funktionsköpfen und optischer Gliederung sowie mit Funktionen und Algorithmen. Die formellen Dinge sind vor allem Fleißarbeit. Es erfordert nicht selten akribische Detektivarbeit, um die Finessen der Funktionen und Algorithmen herauszuarbeiten, wenn hier z.B. alle Tricks der Echtzeit- und Speicheroptimierung eines Assemblers genutzt wurden.

Gleiches gilt für das Ermitteln der ursprünglich gedachten Architektur, die möglicherweise zugewuchert ist wie ein Schrebergarten, der jahrelang keine Pflege mehr genossen hat.

Architekturanalyse: Bohren und Graben

Warum ist die Dokumentation der Architektur so wichtig? Die Architektur ist der Schlüssel zum Verständnis des ganzen Systems und seiner Abhängigkeiten. Nur wer die Architektur kennt, weiß, wie und wo die Eigenschaften des Systems verändert oder erweitert werden können, ohne es in Gefahr zu bringen. Es ist wie der Plan eines Hauses, der zeigt, wo sich tragende Mauern, Rohrleitungen oder Versorgungsleitungen befinden. Zudem gibt die Architektur natürlich auch wichtige Hinweise auf die Schwächen und Grenzen des Systems. Diese Information ist nicht vollständig im Code zu finden und verlässt nicht selten mit dem Wissensträger die Firma.

Stellen Sie sich vor, Sie müssen das Organigramm einer Firma durch die Befragung aller Mitarbeiter über Funktion und Kommunikationswege rekonstruieren. So etwa läuft es, wenn Sie Architektur aus dem vorhandenen Code rekonstruieren wollen. Eine Rekonstruktion mag logisch sein, muss aber noch lange nicht dem ursprünglich beabsichtigten Zweck entsprechen.

Es spricht also alles dafür, dem Systemarchitekten nicht nur Zeit dafür zu geben, etwas zum Laufen zu bringen, sondern auch dafür, die Mechanismen zu dokumentieren. Gute Systemarchitekten sind übrigens sehr gefragt! Sie möchten sicher verhindern, dass er sein Wissen in andere Firmen trägt und Ihre Firma die Unwissenheit mit ins selbstgeschaufelte Grab nimmt.

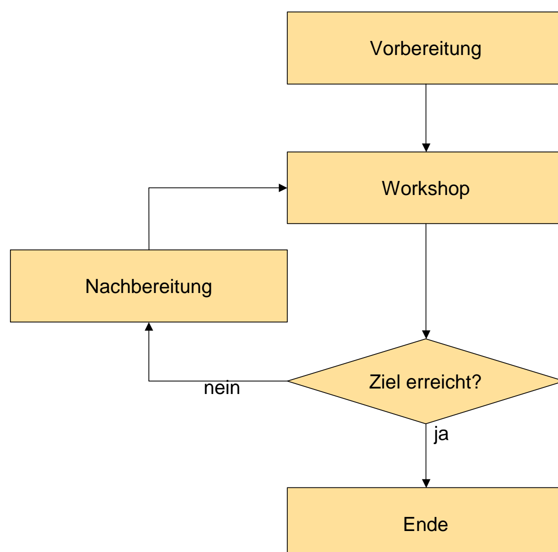

Abbildung 6: Ablauf Architekturanalyse

Der wichtigste und wahrscheinlich schwierigste Teil der Vorbereitung dürfte es sein, einen Termin für alle Wissensträger und beteiligten Entwickler zu finden. Die Auswirkung (Zeitmangel), deren Ursache (fehlende Dokumentation) bekämpft werden soll, steht der Behebung der Ursache meist im Weg – ein Teufelskreis.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Doch Reverse Engineering heißt im übertragenen Sinn waschen, schrubben UND nass werden – mit der Aussicht, danach alles im Reinen zu haben. Wer sich nie richtig wäscht, muss sich nicht wundern, wenn er mit der Zeit streng riecht. Zum Thema Gestank, auf englisch ‚smell', kommen wir übrigens später noch.

Irgendwann muss das Wichtige Vorrang vor dem Dringenden haben. Das ist eine alte Management-Regel, an die Sie Ihre Manager zu gegebener Zeit gerne erinnern dürfen. Bei einer Einladung zum entsprechenden Workshop dürfen auf keinen Fall Angaben zu Ziel, Agenda und Dauer fehlen. Sehr hilfreich ist das Generieren von Reports mit Hilfe von Tools wie Doxygen oder Doc-O-Matic (siehe Info Pool: Tools und Web-Tipps). Mit ihrer Hilfe lassen sich erste Anhaltspunkte zu den inneren Zusammenhängen der Software gewinnen.

Auch die Auswahl eines Dokumentationstools, das sich für die Darstellung von Softwarearchitekturen eignet, will gut bedacht sein. Im einfachsten Fall sind dies Flipcharts und bunte Stifte, doch weit flexibler sind UML-Tools (siehe Info Pool: Tools und Web-Tipps), die einen guten Fundus nützlicher Diagramme bieten. Sie haben auch den Vorteil, dass sie im Großen und Ganzen einem Standard folgen, der sich in der Softwareentwicklung immer stärker durchsetzt. Auch schaffen Sie sich damit eine gute Basis für einen späteren Umbau Ihrer Software-Architektur.

Nun kann der Workshop starten. Als Einstieg empfiehlt es sich, die Software zunächst einmal in Schichten aufzuteilen und diesen die identifizierten Softwareteile zuzuordnen. In Embedded Systemen könnte das so aussehen: Eine Schicht ist die Benutzerschnittstelle mit dem Paket „Human Interface". Dann kommt eine weitere Schicht für alle Steuer- und Regelfunktionen, die der Controller ausführt, mit Paketen wie „Drive Control". Schließlich die Schicht der Hardwareinterfaces mit den Paketen „Sensor- und Aktor-Treiber" und „Kommunikationstreiber".

Im nächsten Bild ist ein solches rudimentäres Architekturbild in einem UML-Diagramm dargestellt.

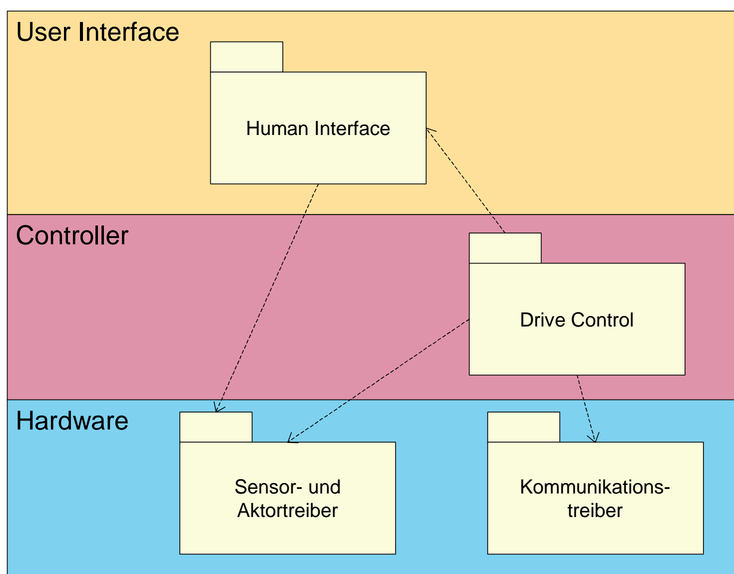

Abbildung 7: Rudimentäres Architekturmodell

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Neben den Paketen sind auch die Abhängigkeiten mit Hilfe der gestrichelten Pfeile dargestellt. Der Pfeil zeigt jeweils in Richtung der Abhängigkeiten. Der Controller braucht offensichtlich sowohl das Benutzer- als auch das Hardwareinterface.

Dieses einfache Beispiel zeigt, wie Sie diese Diagramme nutzen können. Mit Hilfe der Tools lassen sich sehr komplexe Zusammenhänge gut visualisieren. Legen Sie fest, wie umfangreich die UML auch im weiteren Vorgehen eingesetzt wird und welche Perspektiven das Tool bieten soll (Codegenerierung, Repository). Dann wählen Sie das Tool, das entweder schon bei Ihnen im Unternehmen verfügbar ist, oder suchen gezielt ein passendes Werkzeug. Vom mäßig komfortablen Zeichenwerkzeug (Visio) bis zum kompletten Modellierungsbaukasten (ARTiSAN, Rhapsody, etc.) finden Sie je nach Anspruch und Budget heute alles.

Die Projektcoaches von MicroConsult unterstützen Sie hier mit ihrer langjährigen Anwendererfahrung im Embedded Umfeld. Mindestens ein Teammitglied sollte gute Kenntnisse in der Anwendung der Methode und des Tools haben (siehe Info Pool: Training).

Für diese Workshops ist es vorteilhaft, einen unabhängigen Moderator oder Projektcoach mit der Leitung zu betrauen, der vor allem den Ablauf und das Ziel ständig im Auge behält. Auch empfiehlt sich ein unabhängiger Protokollführer, damit sich alle anderen auf die fachlichen Aspekte konzentrieren können.

Die vorbereiteten Reports bieten immer wieder Orientierungshilfen. Gegebenfalls wird der Quellcode inspiziert, um Zusammenhänge abzusichern. Im Verlauf werden Aufgaben mit Terminen festgelegt, die nach dem Workshop zu erledigen sind. Nicht anwesende Personen können mit einbezogen werden. Das wichtigste jedoch ist:

Alle Erkenntnisse müssen SOFORT dokumentiert werden, damit nichts verloren geht.

Am Ende des Workshops sollte festgestellt werden, inwieweit die gesteckten Ziele erreicht wurden und gegebenenfalls ein Folgetermin vereinbart werden.

Ein sehr nützlicher Nebeneffekt des Workshops ist, dass das ganze Team möglicherweise zum ersten Mal eine Vorstellung von der Architektur bekommt.

Codeanalyse: Sieben und Sortieren

Ein tieferer Einstieg in die Software, vor allem bei Assembler und C-Code, zeigt, dass es sich weitgehend um „Handarbeit“ handelt, die nur rudimentär durch Tools unterstützt werden kann. Hilfreich bei der Aufwandsabschätzung sind die Angaben zur Fagan-Inspektion. Bei der Quellcode-Analyse geht man hier von einer Geschwindigkeit von ca. 100 Codezeilen pro Stunde aus.

Es ist offensichtlich, dass man diese Aufgabe nicht einfach nebenher auf die Schnelle erledigen kann. Die notwendigen Zeit muss bei der Zeitplanung der Projekte berücksichtigt werden.

Für die Umsetzung bieten sich Workshops, kleine Arbeitsteams oder koordinierte Einzelmaßnahmen an. Wir empfehlen ein Kick-Off-Meeting, das Ziele und Umfang der Codeanalyse in Verbindung mit einem Zeitplan, Meilensteinen und Verantwortlichkeiten festlegt. Die Ergebnisse werden in den Projektplan bzw. den Zeitplan der Beteiligten aufgenommen. Im Zuge der Vorbereitung werden auch, falls nicht vorhanden, Vorlagen für Datei- und Funktionsköpfe oder andere formelle Konventionen vereinbart.

Bei der Umsetzung empfiehlt sich wieder ein systematisches und schrittweises Vorgehen, das die Dokumentation von übergeordneten zu detaillierten Eigenschaften der Software aufbaut. Man beginnt beispielsweise mit dem Nachtragen der Dateiköpfe und geht dann weiter zu den Funktionsköpfen. Soweit notwendig und zeitlich machbar, wird der Quellcode übersichtlich gegliedert und an Schlüsselstellen kommentiert. Auch hier sind die gewonnenen Erkenntnisse sofort festzuhalten.

Folgende Tools unterstützen bei dieser Tätigkeit:

- Das Freeware-Tool Doxygen (s. Info Pool: Tools und Web-Tipps) extrahiert Informationen, die vorher durch entsprechende Marken (Tags) gekennzeichnet wurden. Es stellt die Abhängigkeiten von Code und Dateiköpfen (Headern) dar und liefert ein Bild der Aufrufhierarchie der Funktionen. Entsprechend aufbereiteter Code kann so recht komfortabel in eine kompakte Dokumentation überführt werden.
- Das kommerzielle Tool Doc-O-Matic bietet eine ähnliche Funktionalität, kann aber nach Angaben des Herstellers auch Funktionsköpfe dokumentieren, die keine speziellen Marken enthalten. Dies wird durch eine intelligente Analyse der Kommentare erreicht. Eine kritische Prüfung der Ergebnisse ist angebracht.

Doppelgänger und Kopierteufel

Codedubletten treten häufig in Programmen auf, die nach dem Copy&Paste-Verfahren erstellt wurden. Sie blähen nicht nur den Code unnötig auf, sondern sind auch eine Quelle für Fehler, die dann möglicherweise übersehen oder aber nicht durchgängig in allen Kopien behoben werden. Tools wie PMD, duplo oder same bieten hier gute Unterstützung, gerade wenn solche Schwächen später beim Refactoring auch korrigiert werden sollen.

Fazit:

- Reverse Engineering gehört in die Projektplanung.
- Der Aufwand lässt sich auf Basis der Richtwerte der Fagan-Codeinspektion abschätzen.
- Umfang und Form der Dokumentation orientieren sich an den gesteckten Zielen und der verfügbaren Zeit.
- Systematisches Vorgehen ist ein Muss – von außen nach innen, vom Wichtigen zum Unkritischen.
- Einige Tools bieten Orientierungs- oder Entscheidungshilfen und erleichtern die künftige Nutzung und Erweiterung der Dokumentation.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Die 3 Rs des Embedded Software Redesigns

Teil III
Refactoring: Aus Alt mach Neu

Die Durchführung des Reverse Engineering liefert in vieler Hinsicht wertvolle Hilfen für die Optimierung der Architektur und des Codes: die Dokumentation des Status Quo, nützliche Tools hierfür, sowie ein Team mit einer klareren Vorstellung von Struktur und Verhalten der Software.

Streng genommen bedeutet Refactoring die Verbesserung der internen Softwarestruktur und Codierung ohne Veränderung des von extern beobachtbaren Verhaltens der Software oder des Systems.

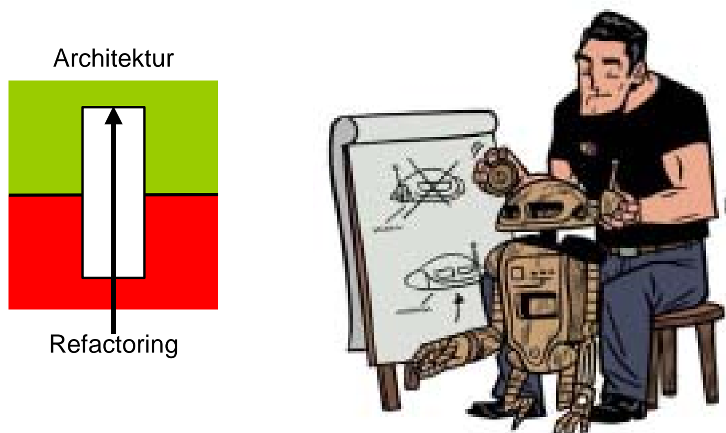

Abbildung 8: Alte Funktion im besseren Design

Die folgenden Voraussetzungen sollten geschaffen werden:

Es muss nachweisbar sein, dass das umgebaute System alle bekannten Funktionen des ursprünglichen Systems ausführen kann, ohne die nun ebenfalls bekannten Schwächen mitzunehmen.

Im Idealfall gibt es einen möglichst automatisch ablaufenden Blackbox-Test, der auch auf die neue Software angesetzt werden kann. Die korrekte Umsetzung muss sich in jedem Fall durch einen Test nachweisen lassen.

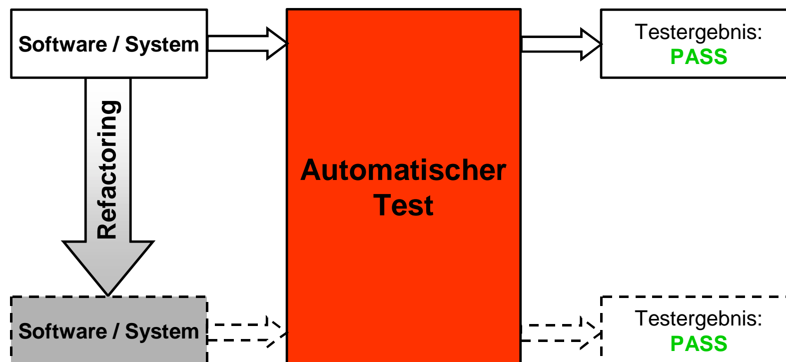

Abbildung 9: Refactoring und Test

Ein Refactoring kann aus den verschiedensten Gründen erforderlich werden. Weit verbreitet ist die Codiermethode vom biologischen in den elektronischen Speicher, sprich vom Hirn in den Computer. Dies führt bei einfachen Systemen zu schnellen Ergebnissen, bei komplexerer Software jedoch ins Chaos. Eine weitere Ursache für Undurchsichtigkeit ist die Entstehung von Seriensoftware aus Prototypen. Der gute Vorsatz, alles noch einmal sauber zu programmieren, wenn der Kunde erst einmal gewonnen ist, wird meist dem Termindruck geopfert. „Es funktioniert ja, noch (!) haben wir die Sache im Griff. Sobald wir Luft haben, holen wir alles nach". Ein nur scheinbar gutes Argument, die Dokumentation fallen zu lassen.

Kommt Ihnen das bekannt vor? Virtuos programmieren sich die Entwickler der ersten Stunde schleichend um Kopf und Kragen. Oder sie verlassen das Unternehmen, um es anderswo von Anfang an richtig zu machen und dieses Damoklesschwert endlich loszuwerden. Die Entwickler, die nachrücken, haben es schwer.

Vielleicht haben Sie aber auch alles richtig gemacht. Doch die Aufgaben und das Umfeld werden immer komplexer. Der richtigen Zeitpunkt ist nicht leicht zu erkennen: Ab wann ist es aufwändiger, mit Altbewährtem weiterzumachen als ungelenk Neues zu wagen? Meist geht es weniger um das Erkennen als die Bereitschaft, der Realität ins Auge zu sehen – anders ausgedrückt: Betriebsblindheit.

Ein gutes Gegenmittel: Lehnen Sie sich mindestens einmal pro Jahr mit Ihrem Team zurück und betrachten Sie dann Ihr Werk und Ihre Arbeitsweise. Sehr hilfreich sind dabei externe Coaches, die diesen „Erkenntnisprozess" anregen und moderieren.

Unabhängig von diesen Motiven gibt es sehr gute Gelegenheiten zum Refactoring, wenn Sie den Stier ohnehin bei den Hörnern packen müssen. Gelegenheiten gibt es viele: Neue Funktionen sollen integriert oder Fehler behoben werden. Eine Zertifizierung steht an und macht Veränderungen zwingend erforderlich.

Das Unternehmen hat sich zu einem Paradigmenwechsel in der Softwareentwicklung entschlossen, beispielsweise in Richtung OOP. Ein RTOS (Real-Time Operating System) wird eingesetzt oder andere zugekaufte Softwareprodukte werden integriert. Die Software soll auf eine neue Hardwareplattform portiert werden; die Portierung auf Multicore-Plattformen stellt eine besondere Herausforderung an die Flexibilität der Softwarearchitektur. Einfachster Grund: Der Kunde verlangt diesen Einsatz, oder der Wettbewerb bietet bereits Lösungen an. Nur wer eine Softwarearchitektur bauen kann, die sich auf verschiedene Plattformen portieren und um künftige Anforderungen erweitern lässt, hat auf Dauer gute Chancen, die Innovationszyklen für sich zu nutzen.

Ehe Sie sich auf Architektur und Code stürzen, ist es wichtig, das **Ziel** des Refactorings zu klären:

- Bessere Wartbarkeit
- Einfachere Erweiterbarkeit
- Bessere Verständlichkeit
- Besseres Zeitverhalten
- Geringerer Speicherverbrauch
- Höhere Zuverlässigkeit
- Höhere Sicherheit
- Plattform-Unabhängigkeit
- Kürzere Projektdauer
- Kürzere Einarbeitungszeit
- etc.

Beim Betrachten dieser Punkte wird schnell klar, dass es eher darum geht, mehrere Ziele sinnvoll zu optimieren als ein Ziel zu maximieren. Auf jeden Fall sollte das übertriebene plattformabhängige Optimieren von Code auf minimalen Speicher oder minimale Laufzeit kritisch hinterfragt werden, wenn es zu unflexiblen und schwer verständlichen Ergebnissen führt, denn damit steigt das Fehlerrisiko.

(Eine kleine boshafte Frage an die Automobilindustrie: Bringt die Verkleinerung des Arbeitsspeichers eines Steuergerätes zur Senkung der Stückkosten wirklich mehr Einnahmen als die dadurch entstehenden zusätzlichen Entwicklungskosten? Wurde hierbei auch berücksichtigt, dass durch die jetzt einsetzende Bastelei die Wahrscheinlichkeit einer sehr kostenintensiven Rückrufaktion steigt?)

Gründe und Ziele gibt es also genügend.

Bei der Umsetzung unserer Ziele unterscheiden wir zwischen dem großen und den kleinen Refactoring. Thema des **großen Refactoring** ist die Architektur jenseits von Funktionen, Daten und Klassen. Es geht darum, eine Architektur einzuführen, falls bisher noch keine vorhanden ist, oder darum, die Architektur zu verbessern, um die zuvor definierten Ziele auf Architekturebene zu erfüllen. Das **kleine Refactoring** betrifft die Ebene des Quellcodes oder der Klassen. Fangen wir zunächst klein an.

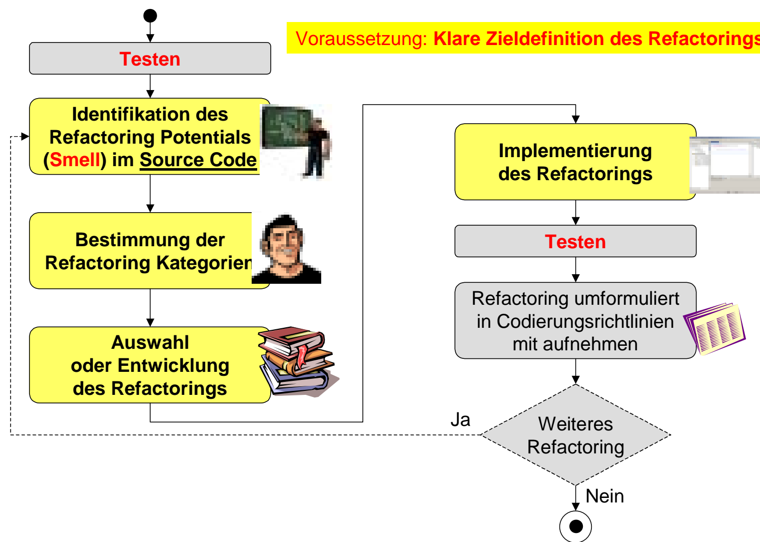

Abbildung 10: Ablauf des kleinen Refactorings

Ein speziell erstellter oder vorhandener Test bestätigt das Verhalten der bestehenden Software. Dann wird das Refactoring-Potential identifiziert. Hier lässt sich gut auf den Ergebnissen des Reverse Engineering aufbauen. Im Englischen spricht der Entwickler von sogenannten „smells“ – frei übersetzt: „Wo stinkt es in der Software?“. Nun ist festzulegen, welches Ziel mit dem Refactoring angestrebt wird und welche Refactoring-Kategorien zum Einsatz kommen. Eine Kategorie ist die Beschreibung der prinzipiellen Maßnahmen zur Erreichung des Ziels. In jeder Refactoring-Kategorie sind mehrere mögliche Refactorings beschrieben. Das Refactoring selbst beschreibt die konkrete Umsetzung auf der Basis von Source-Code.

Beispiel[1]:

Zieldefinition:	Source-Code-Optimierung zur Erhöhung der Wiederverwendbarkeit
Refactoring-Kategorie:	Composing Method In dieser Kategorie sind Refactorings zum Umcodieren von Funktionen gelistet.
Refactoring:	Extract Method Ein mögliches Refactoring zum Umcodieren von Funktionen ist die Möglichkeit, eine neue zusätzliche Funktion zu generieren und diese in der bereits bestehenden aufzurufen.
Refactoring-Kategorie:	Simplifying Conditional Expressions In dieser Kategorie sind Refactorings zum Vereinfachen bedingter Ausdrücken gelistet.
Refactoring:	Decompose Conditional Bedingte Ausdrücke lassen sich vereinfachen, indem die Bedingungsabfrage und jeder mögliche Bearbeitungszweig über einen Funktionsaufruf ausgeführt werden.

[1] Refactoring: Improve the Design of Existing Code, Martin Fowler, Addison-Wesley, ISBN 0-201-48567-2

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Viele bereits bekannte Refactorings lassen sich in Editoren oder durch ein Plug-In automatisch durchführen (siehe Info Pool: Tools und Web-Tipps). Nach Durchführung des Refactorings wird getestet, ob das Verhalten der Software erhalten bzw. das gewünschte Verhalten erreicht werden konnte.

Hier noch ein paar Tipps:

- Strukturieren Sie Ihre Software so durchsichtig wie möglich.
- Wählen Sie möglichst aussagekräftige Namen für Variable, Prozeduren, Klassen und Methoden. Die Software wird damit auch ohne Kommentar weitgehend verständlich.
- Halten Sie sich nicht nur an vorgegebene Kataloge von Refactorings, sondern entwickeln Sie eigene.
- Passen Sie die Programmierrichtlinien auf Basis der Erkenntnisse an.

Im Prinzip läuft das große Refactoring genauso ab. Bei komplexer Software wird jedoch inkrementell vorgegangen und zunächst einmal auf Architekturebene gearbeitet.

Üblicherweise schließt sich das zuvor beschriebene kleine Refactoring an: Wir haben es also mit einer Kombination aus kleinem und großem Refactoring zu tun. Auch hier werden die Erkenntnisse in neuen Architektur- und Codierungsrichtlinien festgehalten.

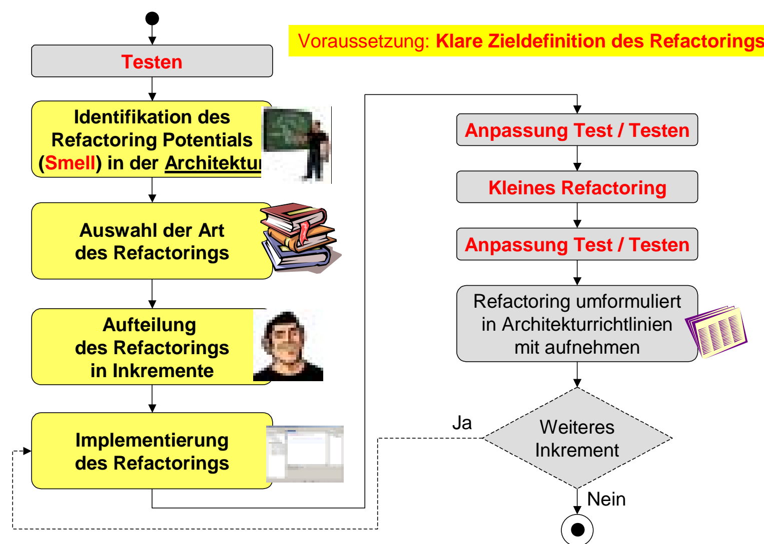

Abbildung 11: Ablauf eines großes Refactorings

Beispiel:

Zielsetzung ist die zukünftige Verbesserung von Wiederverwendbarkeit, Wartbarkeit und Erweiterbarkeit. Ein mögliches Refactoring könnte die objektorientierte Modellierung mit der UML und die anschließende Codierung in C sein.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Die Ausgangssituation ist das dargestellte, prozedural in C codierte Counter-Modul:

```
int count;
int min;
int max;

void Counter_init(int par_count, int par_min, int par_max)
{
	count = par_count;
	min = par_min;
	max = par_max;
	Counter_print();
}

void Counter_counting(void)
{
	if (count >= min && count < max)
	{
		count = count + 1;
		Counter_print();
	}
	else
		count = 0;
	}

void Counter_print(void)
{
	printf("Min value = %i ", min);
	printf("Max value = %i ", max);
	printf("Count value = %i\n", count);
}
```

Mit dem objektorientierten Ansatz ergibt sich dieses UML-Klassendiagramm:

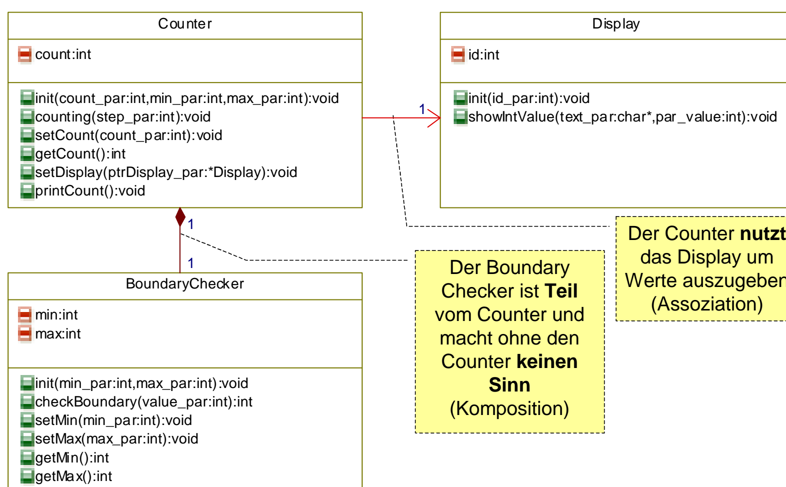

Abbildung 12: Klassendiagramm zum Counter

Wenn Sie das Softwaremodell und den dazugehörigen Source-Code konsistent halten, liefert Ihnen die UML-Darstellung gleichzeitig die Dokumentation des Source-Codes. Die grafische UML-Darstellung macht gleichzeitig den Source-Code auch für neue Projektmitglieder besser verständlich. Auch hier können die bereits erwähnten UML-Tools zum Einsatz kommen (s. Info Pool: Tools und Web-Tipps).

Da Entwicklungsumgebungen für die PC-Welt (z.B. integriertes Refactoring) meist leistungsfähiger sind als Entwicklungsplattformen für Mikrocontroller, sollte die PC-Entwicklungsumgebung auch für Embedded Software so lange wie möglich genutzt werden.

Fazit:

- Refactoring ist Teil des Entwicklungsprozesses und der Projektplanung.
- Erfolgreiches Refactoring erfordert eine gute Dokumentation der vorhandenen Software.
- Refactoring kann auf Architektur- und Quellcodeebene ansetzen.
- Jedes Refactoring hat ein Ziel, das im Auge behalten werden muss.
- Nach dem Refactoring werden ggf. die Richtlinien für Codierung und Architektur angepasst.
- Nur ein Test vorher und nachher macht die Ergebnisse nachvollziehbar.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Die 3 Rs des Embedded Software Redesigns

Teil IV
Reengineering: Der Weg ist das Ziel

Der Weg ist das Ziel – das klingt nicht gerade nach einer Weisheit aus der Softwareentwicklung. Große Weisheiten haben jedoch den Vorteil, dass sie technologieunabhängig und zeitlos sind. Die Notwendigkeit, durch die Müh(l)e von Reverse Engineering und Refactoring zu gehen, liegt viel mehr im Weg als im Ziel begründet.

Das Ziel ist eine lauffähige, möglichst fehlerlose Software. Durch ein **Reengineering** des Entstehungsprozesses lässt sich dieses Ziel künftig schneller und einfacher erreichen. Dabei sollte man sich vergegenwärtigen, dass nach dem Projekt immer auch vor dem Projekt ist.

Um bei unserem Bild zu bleiben: Für aufgenommene Kredite in einem Projekt muss man in der Regel spätestens in den nächsten Projekten die Schulden samt Zinsen zurückzahlen. Auch wenn die Versuchung groß ist, bedeutet schneller nicht unbedingt besser, gerade wenn es um die Basis für künftige Produktgenerationen geht.

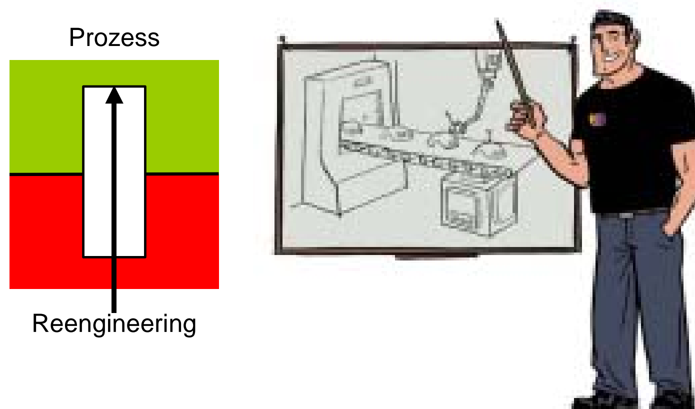

Abbildung 13: Prozessverbesserung

Bild 14 zeigt typische Elemente eines Prozesses. Er besteht aus sogenannten Core-Workflows oder Kernaufgaben, wie Anforderungsanalyse oder Implementierung, die den Weg von der Idee bis zum fertigen Programm begleiten. Der Weg unterteilt sich in Phasen, die durch Meilensteine terminiert sind.

Auf diese Weise entstehen definierte Synchronisationspunkte und Zwischenergebnisse, die eine Einschätzung des Projektfortschritts und weiteren Verlaufs ermöglichen.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Die Darstellung ist realitätsnäher als das Wasserfall- oder V-Modell, denn sie zeigt, dass jeder Workflow das gesamte Projekt begleitet, wenn auch mit unterschiedlicher Intensität. Es wird beispielsweise nicht erst am Ende der Implementierung getestet. Vielmehr werden, sofern möglich, immer wieder Zwischenergebnisse überprüft.

Neben den Core-Workflows gibt es noch die Supporting Workflows, wie die Bereitstellung der Infrastruktur oder die Projektleitung. Sie betreffen die Produktentwicklung zwar nicht direkt, sind jedoch für einen reibungsarmen Ablauf von großer Bedeutung.

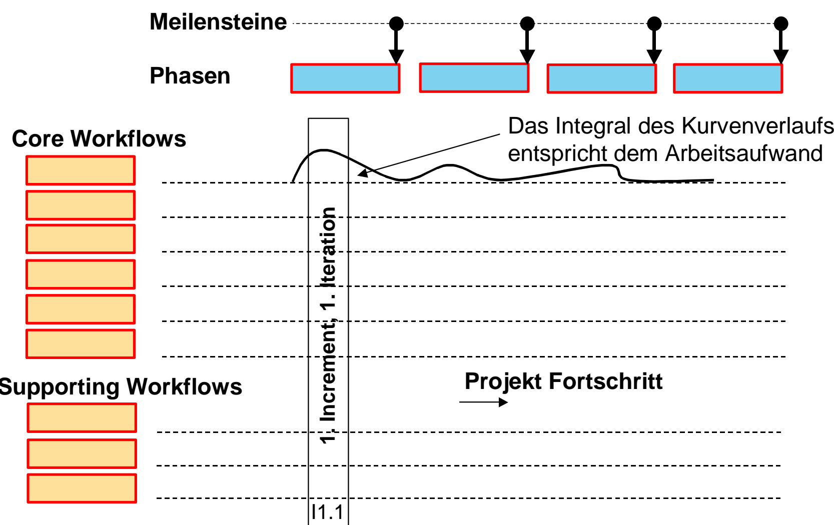

Abbildung 14: Allgemeines Prozessschema

Moderne Prozessmodelle ermöglichen ein inkrementelles und iteratives Vorgehen. Inkrementell bedeutet, dass das Gesamtprojekt in Teilprojekte (Inkremente) aufgegliedert wird. Die Inkremente durchlaufen bis zur Integration unabhängig voneinander die im Prozesses definierten Workflows. Die Inkremente können parallel und/oder seriell zueinander entstehen.

Besonders in der Softwareentwicklung gibt es heute nur sehr begrenzte Möglichkeiten, basierend auf einer Modellsimulation definierte Aussagen über die Ausführbarkeit auf dem Target zu treffen. Hier kommt das iterative Vorgehen ins Spiel.

Beispiel: Bei der Ausführung eines Inkrements auf dem Target wird ein Fehlverhalten festgestellt. Der Entwickler ermittelt, in welchem der zuvor durchlaufenen Workflows dieses Fehlverhalten seine Ursache hat. In diesem Workflow korrigiert der Entwickler die Ursache des Fehlverhaltens und durchläuft nochmals alle folgenden Workflows mit seinem Inkrement. Der ersten Iteration können sich weitere anschließen, bis das Fehlverhalten vollends beseitigt ist.

Gerade bei Projekten mit neuen Technologien, Tools oder Methoden erlauben die Iterationen über alle Workflows in jeder Projektphase das Einfließen wichtiger Erfahrungen in weitere Iterationen.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Der Workflow selbst beschreibt, durch welche Aktivitäten Ausgangsartefakte aus sogenannten Eingangsartefakten entstehen und welche Rollen daran beteiligt sind. Ein Artefakt kann zum Beispiel sein: Hardware, Datenbuch, Dokumentation, Testprotokolle, Datenbankeinträge, etc. Im Workflow beschreiben Aktivitäten, wie die Artefakte verarbeitet werden. Für jede Aktivität sind Methode und Notation beschrieben, und für jedes Artefakt sind Regeln (Guidelines) definiert, wie dieses auszusehen hat. Handelt es sich bei den Artefakten um Dokumente, beschreiben die Vorlagen (Templates) das Aussehen.

Beispiele für Rollen sind Entwickler, Tester, Projektleiter oder Einkäufer. Eine Rolle kann, aber muss nicht, einer Person zugeteilt sein. So ist es bei Projekten in kleineren Firmen nicht ungewöhnlich, dass Projektleiter, Entwickler und Tester in einer Person vereint sind. Allerdings ist die Vereinigung der Rollen Tester und Entwickler wegen des offensichtlichen Rollenkonflikts meist problematisch.

Jeder Rolle werden neben den Aufgaben die erforderlichen Fähigkeiten und das notwendige Wissen zugeschrieben. Diese Definitionen sind für personelle Entscheidungen sehr hilfreich.

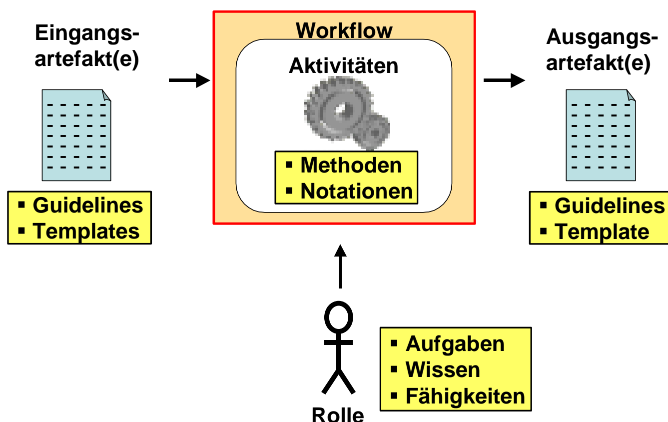

Abbildung 15: Workflow

Alle Beschreibungen werden in der Prozessdokumentation zusammengefasst. Jeder Entwickler oder jedes Entwicklungsteam hat natürlich so etwas wie einen Prozess, nach dem man vorgeht. Nicht selten jedoch sind diese Prozesse hochgradig intuitiv (manche sagen auch boshaft: chaotisch) und unzureichend oder gar nicht dokumentiert. Bei sehr kleinen Teams und relativ geringer Komplexität mag diese Vorgehensweise noch gut funktionieren und möglicherweise sogar schneller zum Ziel führen. Wenn Aufgaben aber immer wieder doppelt und oder gar nicht bearbeitet werden, sind die Grenzen schnell erkennbar.

Dokumentation hat zudem auch einen klaren Vorteil: Dokumentieren heißt Reflektieren, und Reflektieren bedeutet, Verbesserungspotential erkennen. Natürlich kann man alles übertreiben, doch die Regel ist eher die Untertreibung.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Spätestens bei einer Zertifizierung des Prozessmodells – ob nach ISO (International Organization for Standardization), SPICE (Software Process Improvement and Capability Determination) oder CMMI (Capability Maturity Model Integration) – heißt es Farbe bekennen.

Prozesse, die gleichzeitig auch Lernprozesse sind, werden sich mit der Zeit verändern und weiterentwickeln. Es spricht also vieles dafür, Prozesse von Anfang an so zu beschreiben, dass Schwächen schnell erkannt werden und Änderungen leichter möglich sind.

Wir empfehlen dazu aus mehreren Gründen die UML:

- UML ist ein Industriestandard.
- UML ist in vielen Unternehmen verbreitet.
- UML wird durch Case Tools unterstützt.
- Das Prozessmodell lässt sich durch die Nutzung von UML einfach adaptieren.
- Prozessdokumentation kann automatisch in verschiedenen Formaten (z.B. Word, PowerPoint und HTML) aus dem UML-Modell generiert werden.
- Die Prozessdokumentation ist mit individuellen Templates skalierbar.
- Der Inhalt der Prozessdokumentation kann aus dem UML-Modell ausgewählt werden. Somit lassen sich für verschiedenen Rollen unterschiedliche Prozesssichten generieren.

MicroConsult kennt die Vorteile aus eigener Erfahrung: Wir haben einen Musterprozess für die Embedded Entwicklung auf Basis der UML entwickelt, der sich sehr schnell an die Projekterfordernisse unserer Kunden anpassen lässt.

Dabei haben wir die zahlreichen publizierten Prozesse – V-Modell, USDP (Unified Software Development Process), RUP (Rational Unified Process), MSF (Microsoft Solution Framework), XP (Extreme Programming), OE (Object Engineering), ROPES (Rapid Object Oriented Process for Embedded Systems), COMET (Concurrent Object Modeling and Architectural Design Method) oder Octopus (Object-Oriented Technology for Real-Time Systems) – mit unseren eigenen Erfahrungen aus der Embedded Entwicklung zu dem Prozessmodell **COPES (Customizable Object Oriented Development Porcess for Embedded Systems)** kombiniert.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

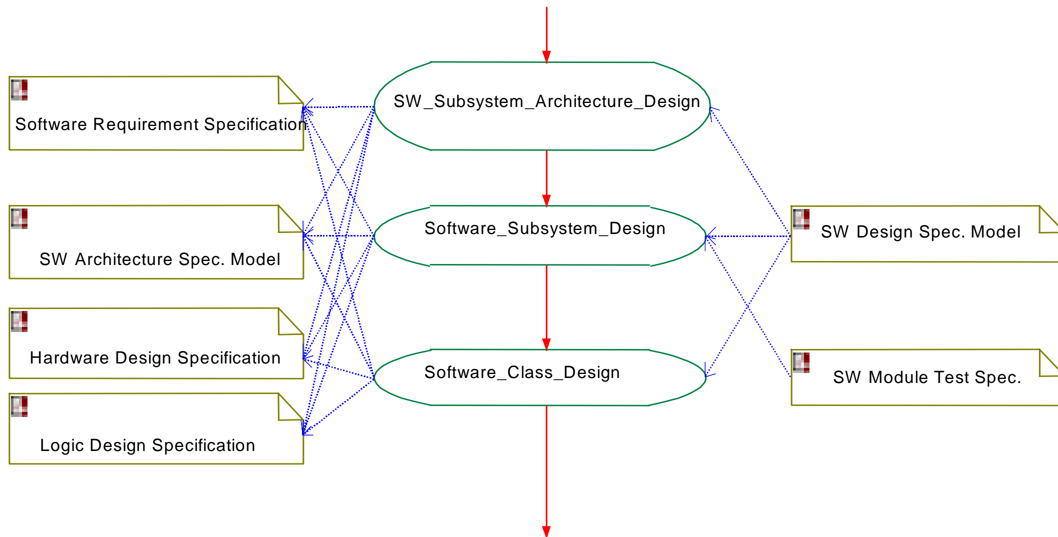

Abbildung 16: Ausschnitt COPES - Workflow Software Design (Aktivitätsdiagramm aus der UML)

Bild 16 zeigt einen Ausschnitt aus diesem Modell. Links sehen Sie die Eingangs- und rechts die Ausgangsartefakte. In der Mitte sind drei Arbeitsschritte beschrieben. Die blauen gestrichelten Pfeile beschreiben die Abhängigkeiten der Modellelemente untereinander.

Der Pfeil zeigt jeweils in Richtung der Abhängigkeiten. Die roten durchgehenden Pfeile zwischen den Arbeitsschritten sind als Vorgehensrichtung zu verstehen.

Das Elegante an dieser Beschreibung ist, dass – falls mit UML entwickelt wird – Prozess und Software mit der gleichen Notation beschrieben werden können und über das gleiche Tool verfügbar sind – ein zusätzlicher Nutzen des UML-Tools.

Wir haben unser Prozessmodell COPES aufgrund unserer Erfahrungen um das Thema **Software Redesign** erweitert.

Auf den nächsten Bildern sehen Sie die von uns vorgesehene Rollenverteilung. Diese Prozessmodelle sind ein anpassbares Framework, das sich in der Praxis bereits bewährt hat.

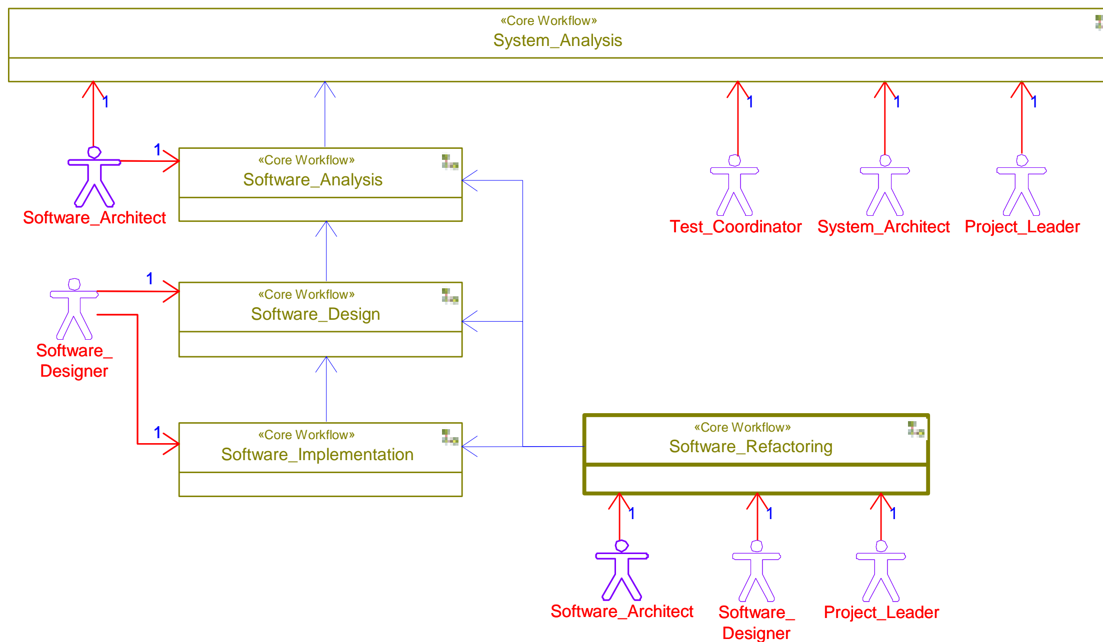

Abbildung 17: Ausschnitt COPES:
Refactoring Workflow-Integration mit Rollenbeschreibung (Klassendiagramm aus der UML)

Die **Prozessdokumentation** sollte so erstellt oder angepasst werden, dass alle wichtigen Rollen dargestellt sind, auch wenn in kleineren Teams mehrere Rollen durch eine Person gespielt werden. Das schafft die Grundlage für das Erkennen von Rollenkonflikten oder Ansätze für effektivere Arbeitsteilung. Nur ein lebbarer Prozess, von der Mehrzahl der Beteiligten getragen, verspricht Erfolg. Deshalb ist der Konsens hier unverzichtbar, ganz abgesehen von dem fruchtbaren Meinungs- und Erfahrungsaustausch in der Findungsphase.

Mit Hilfe eines UML-Modells wie COPES und der Unterstützung erfahrener Projektcoaches lässt sich dieses Ziel oft innerhalb weniger Tagen erreichen.

Fazit:

- Prozessbeschreibungen sind hilfreich und notwendig.
- Eine toolbasierende Modellierung des Prozesses bietet große Vorteile.
- Prozessbeschreibungen sollten von den Beteiligten mitgestaltet werden.
- Vorbereitete Modelle wie COPES erleichtern die Prozessdefinition.
- Projektcoaches ersparen Umwege auf dem Weg zum lebbaren Prozess.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Resümee:

Eine regelmäßige und kritische Bewertung von Dokumentation, Programmierung, Architektur und Prozess ist unverzichtbar, um Schwächen rechtzeitig zu erkennen und abzustellen. Dies sollte geschehen, bevor unheilvolle Kredite auf die Software-Qualitätskonten aufgenommen werden. Unterstützung von außen bietet hier entscheidende Vorteile. Wir leisten gerne durch Information, Training, Coaching oder Engineering unseren Beitrag zu Ihrem erfolgreichen Software Redesign.

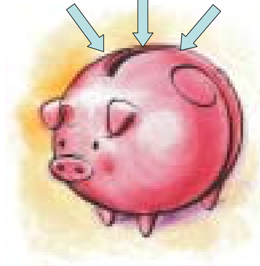

Abbildung 18: Guthaben für Ihr Qualitätskonto

Info-Pool

Buch-Tipps

Refactoring. Improving the Design of Existing Code
Autor: Martin Fowler
Verlag: Addison-Wesley
ISBN 0-201-48567-2
www.refactoring.com

Refactoring to Patterns
Autor: Joshua Kerievsky
Verlag: Addison-Wesley
ISBN 0-321-21335-1

Refactorings in großen Softwareprojekten:
Komplexe Restrukturierung erfolgreich durchführen
Autoren: Stefan Roock, Martin Lippert
dpunkt.verlag
ISBN 3-89864-207-0

Die UML-Kurzreferenz 2.0 für die Praxis
Autor: Bernd Oestereich
Verlag: Oldenbourg
ISBN 3-486-57788-3

UML 2 glasklar
Autoren: M. Jeckle, C. Rupp, J. Hahn, B. Zengler, S. Queins
Verlag: Oldenbourg
ISBN 3-446-22575-7

Real Time UML (Third Edition)
Autor: Bruce Powel Douglass
Verlag: Addison-Wesley
ISBN 0-321-16076-2

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Info-Pool

Tools und Web-Tipps

UML Case Tools:

ARTiSAN Software Tools: ARTiSAN Studio
www.artisansw.com

Telelogic: Rhapsody
www.telelogic.com

Sparx Systems: Enterprise Architect
http://www.sparxsystems.com/

Editoren und Plug-Ins für Refactoring:

Ideat Solutions: Ref++
www.refpp.com

Xref-Tech: Xrefactory for C++
http://www.xref-tech.com/

SlickEdit: SlickEdit
www.slickedit.com

Eclipse Organisation: Eclipse
www.eclipse.org

Finden von Codedubletten:

Freeware: PMD
http://pmd.sourceforge.net/

Freeware: duplo
http://sourceforge.net/projects/duplo

Freeware: same
http://sourceforge.net/projects/same

Codedokumentation:

Freeware: Doxygen
www.doxygen.org

toolsfactory software: Doc-O-Matic
http://www.toolsfactory.de/

Laufend topaktuelle Infos zum Thema Embedded Software Engineering:

www.ese-report.de

MicroConsult hat keinerlei Einfluss auf die Inhalte und Funktion der hier genannten Links.
Die Verantwortung liegt ausschließlich bei den Anbietern dieser Seiten.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

MicroConsult Leistungen

Training zum Thema – die optimale Ausrüstung für jede Tiefe

Mit den MicroConsult-Trainings schaffen Sie sich die optimale Wissensbasis, um Projektanforderungen gerecht zu werden und an Sie gestellte Erwartungen zu übertreffen. Erfahrene Spezialisten vermitteln Ihnen dabei auch komplexe Inhalte verständlich und praxisbezogen und sichern Ihnen den effektiven Umgang mit der richtigen Ausrüstung.

Aktuelle Termine und unser komplettes Trainingsprogramm finden Sie unter www.microconsult.de.

Requirements Engineering und Management für die Entwicklung in der Industrie

Trainingsziel:
Den Requirements Prozess verstehen, bewerten, in der Firma einführen und dort leben; Optimieren der bestehenden Prozesse.

Vorkenntnisse: Keine; Projekterfahrungen sind von Vorteil.

Inhalt:
Entwicklungsprozess; Identifikation von Requirements; Dokumentation anhand von Beispielen und mit grafischen Hilfsmittel der UML; Abnahmetests; Management; praktische Übung anhand eines Beispiels für ein typisches Entwicklungsszenario.

Dauer: 4 Tage

UML Grundlagen: Objektorientierte Analyse mit der UML

Trainingsziel:
Analyse und Entwurfsverfahren sowie die Darstellungsform der Unified Modeling Language (UML) kompetent einsetzen.

Vorkenntnisse: Programmiererfahrung, z.B. CHILL, C, Fortran oder C++.

Inhalt:
Grundbegriffe der OOP; Einführung in die objektorientierten Basiskonzepte; Darstellung der objektorientierten Basiskonzepte mit Hilfe der UML-Klassennotation; dynamisches Verhalten objektorientierter Software; Umsetzung in verschiedene Programmiersprachen.

Dauer: 5 Tage

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

MicroConsult Leistungen

Objektorientierte Analyse für Embedded Systeme

Trainingsziel:
Den Einsatz der UML für Embedded Systeme richtig einschätzen. Systematisches Entwickeln von Softwaresystemen, von der Requirementanalyse bis zum Design.

Vorkenntnisse: Projekterfahrung mit Embedded Systemen und Kenntnisse wie im Training „UML Grundlagen: Objektorientierte Analyse mit der UML" vermittelt.

Inhalt:
UML und Embedded Systeme; Entwicklungsprozess; Requirementanalyse; Realtime Analyse; Design; praktische Übungen (z.B. Entwicklung einer Embedded Applikation mit der UML; Einsatz eines typischen Entwicklungsprozesses).

Dauer: 5 Tage

Design Pattern

Trainingsziel:
Effizientes Einsetzen der wichtigsten Design Pattern im Embedded Bereich.

Vorkenntnisse:
Erfahrung in einer objektorientierten Sprache.

Inhalt:
Einführung in das objektorientierte Paradigma und die UML als Grundlage für die Design Pattern; Delegation und Polymorphie; Erzeugungsmuster; Verhaltensmuster; Strukturmuster; Praxisbeispiele für den Einsatz von Design Pattern im Embedded Bereich; Zusammenwirken mehrerer Muster; Überblick Architekturmuster.

Dauer: 4 Tage

Software-Qualität für Embedded Systeme

Trainingsziel:
Die angestrebte Qualität von Software systematisch vorgeben; Standards zur Qualität von Software kennen; die erreichte Qualität bestätigen.

Vorkenntnisse:
Projekterfahrung mit Softwaresystemen.

Inhalt:
Qualität von Entwicklungsprozessen; Qualität der Produkte Rechner und Software; Bestätigung der Produktqualität; praktische Übungen, z.B. Prüfen von Code oder Bewerten von Prüfergebnissen.

Dauer: 5 Tage

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

MicroConsult Leistungen

Software-Projektmanagement

Trainingsziel:
Methoden und Vorgehensweisen in Entwicklungsprojekten sowie Optimierung des eigenen Projektes.

Vorkenntnisse: Projekterfahrung.

Inhalt:
Projektmanagement; Projektorganisation; bewährte Vorgehensmodelle im Produktentwicklungsprozess; erfolgreicher Projektstart; realistische Planung von Produktentwicklungsprojekten;, Techniken und Prozesse des Projektcontrollings; Projektsimulation mit SimulTrain®.

Dauer: 5 Tage

Software-Test: Strukturiertes und effizientes Testen von Embedded Systemen

Trainingsziel:
Professionelles Testen beherrschen; Tests planen, bewerten und implementieren.

Vorkenntnisse:
Grundkenntnisse einer höheren Programmiersprache, z.B. C/C++.

Inhalt:
Bedeutung von Test im Embedded Markt; Software-Qualität für Embedded-Systeme; Requirementanalyse; Testspezifikation und Testplanung; statische Prüfung vom Review zum LINT; Integrationsstrategien; dynamische Prüfung; Testauswertung; kosteneffiziente Tests.

Dauer: 5 Tage

Embedded C++: Objektorientiertes Programmieren von Mikrocontrollern mit EC++

Trainingsziel:
Sie können die Vorteile der objektorientierten Methode für Embedded Systeme nutzen und deren Nachteile vermeiden.

Vorkenntnisse:
Erfahrung in der Programmierung mit C. Mikroprozessor-Grundkenntnisse sind von Vorteil.

Inhalt:
Einführung in die objektorientierte Begriffswelt; Objektorientiertes Programmieren in C; Einführung in die Programmiersprache C++ (EC++); C++ für Embedded Applikationen; Einführung in die Analyse mit der UML; praktische Übungen (C und C++ Programmierung im OOP-Kontext, Modellierung einer Embedded Applikation mit der UML, Einsatz eines Embedded Targets).

Dauer: 5 Tage

C++ für Fortgeschrittene

Trainingsziel:
Anwendung von Templates, Exceptions und der Standard Template Library (STL); Realisierung fortgeschrittener objektorientierte Konzepte mit C++.

Vorkenntnisse: Kenntnisse wie im Training "OOP mit C++" vermittelt.

Inhalt:
Ausnahmebehandlungen (Exceptions); Standard-Klassen und Funktionen; Templates und Standard Template Library (STL); Hintergrundwissen über C++; konkretes Resource Management; fortgeschrittene Konzepte der Objektorientierung.

Dauer: 5 Tage

C für Fortgeschrittene

Trainingsziel:
Professionelle Anwendung von ANSI-C Bibliotheken und selbstdefinierter Datentypen.

Vorkenntnisse:
Solide C-Kenntnisse wie im Training "Programmiersprache C" vermittelt, oder vergleichbare Programmiererfahrungen in C.

Inhalt:
Sprachelemente (Objekte, Objekttypen, Unions, Strukturen, Datentypen, Typumwandlung); Programmiertechniken (Objektorientierte Programmierung, rekursive Funktionen, komplexe Typen); C-Standardbibliothek (dynamische Speicherverwaltung, Listen-/Blockoperationen, Dateiverarbeitung, Signalbehandlung).

Dauer: 5 Tage

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

MicroConsult Leistungen

Gerne bringen wir auch unser Wissen direkt in Ihr Projekt ein. Mit Coaching unterstützen wir die Lern- und Veränderungsprozesse direkt im Projekt. Oder wir übernehmen Projektaufgaben und entlasten Sie damit.

Coaching – Entscheidende Impulse im rechten Moment

Als fachkundige Experten stehen wir Ihnen in jeder Phase Ihres Projektes zur Seite und betreuen Ihre Entwickler- und Testteams kompetent und effizient. Mit bewährter Methodik und hohem Praxisbezug machen wir bereits im Vorfeld auf Entwicklungs- und Testrisiken aufmerksam.

Somit können Sie Ihr Projekt transparent, vorhersehbar und wirtschaftlich gestalten. Sie reduzieren die Entwicklungszeit auf das Wesentliche und minimieren die Kosten. Die Entwicklung wird in alle Richtungen abgesichert und zielt auf einen qualitativ hochwertigen Projekterfolg.

Engineering – Verstärkung für Projektteams

Wenn Projekte die eigenen Kapazitäten überschreiten, nehmen die Experten von MicroConsult die Last von Ihren Schultern. Wir bieten Ihnen die optimale Unterstützung, damit Sie Ihre eigenen Kräfte dort konzentrieren können, wo sie die größte Wirkung erzielen.

Was nicht zu den Kernkompetenzen Ihres Unternehmens zählt, übertragen Sie unseren Entwicklungsspezialisten: von der Definition und Optimierung von Entwicklungsprozessen über die Entwicklung von Hardware, Software und Systemarchitekturen bis zum Testen des fertigen Produktes.

Die Autoren

Thomas Batt
bringt seit 1999 sein breites Fachwissen in unser Team ein. Als Trainer und Projektcoach hat er sich unter anderem auf die Themengebiete Softwareentwicklung (Tools, Methoden, Prozesse) für Embedded Systeme und auf Echtzeitbetriebssysteme spezialisiert. Hier profitieren unsere Kunden auch von seiner langjährigen Erfahrung als Trainer für Mikrocontrollerplattformen.

Frank Listing
verstärkte uns nach 10 Jahren Software-Entwicklungspraxis. Seit 2002 ist er Trainer und Projektcoach mit dem Schwerpunkt Microsoft-Plattformen, objektorientierte Programmierung und Testen von Embedded Systemen. Er ist fachlich für das Thema .NET verantwortlich und gibt sein Wissen immer wieder in Publikationen und Fachvorträgen weiter. Der beste Ausgleich zu der täglichen Gehirnakrobatik ist für ihn Fahrradfahren und Skaten, und das möglichst schnell.

Peter Siwon
hat die "Embedded Welt" schon aus vielen Blickwinkeln erlebt: In der Forschung, als Trainer, als Entwickler, als Projektleiter, in Vertrieb und Marketing. Heute ist er Geschäftsführer bei MicroConsult. Seit nunmehr 20 Jahren schreibt er regelmäßig Artikel in Fachmagazinen und wirkt als Moderator oder Referent bei Veranstaltungen der "Embedded Welt" mit. Wasser ist sein Element: Er ist leidenschaftlicher Schwimmer und Paddler und trinkt ganz untypisch für einen Bayern am liebsten Wasser.

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH

Impressum

Herausgeber:

MicroConsult GmbH
Charles-de-Gaulle-Str. 6
81737 München
Tel. +49 89 450617-0
Fax +49 89 450617-17
www.microconsult.de

Projektleitung und Redaktion:

Sabine Pagler
Marketing Coordinator
MicroConsult

Autoren:

Peter Siwon
Thomas Batt
Frank Listing
MicroConsult

Trend Guide Medienpartner:

www.elektronikpraxis.de

MicroConsult Microelectronics Consulting & Training GmbH
Charles-de-Gaulle-Straße 6 • D-81737 München
Tel. 089 450617-71 • Fax 089 450617-17
E-Mail: info@microconsult.de • www.microconsult.de

© 2007 MicroConsult – Microelectronics Consulting and Training GmbH