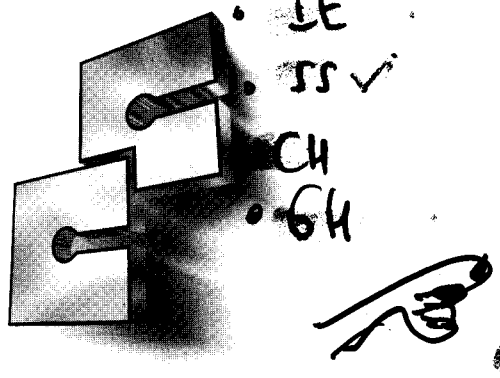

Software Engineering Newsletter

Case Consult — Heft 58

In dieser Ausgabe:

Vorwort

Liebe Leserinnen und Leser,

dieser Newsletter wendet sich an alle, die mit der Entwicklung, Prüfung und Test sowie der Wartung, Weiterentwicklung und Wiederverwendung von Anwendungssoftware zu tun haben. Sie werden in dieser Ausgabe hoffentlich einige Anregungen zu Ihrer Arbeit finden, vor allem was die Integration der vorhandenen Systeme anbetrifft. Die Extensible Markup Language (XML) wird künftig einen großen Einfluß auf Ihre Arbeitswelt haben, egal ob Sie in Assembler, COBOL, C++ oder Java programmieren. XML wird die universale Datenbeschreibungssprache über alle Programmiersprachen hinweg. Programme sind vergänglich, Daten sind unsterblich. Daher ist die Beschreibung der Daten so wichtig. Sämtliche Datenstrukturen, ob Datenbanken, Dateien, Berichte, Webseiten oder interne Nachrichten, werden früher oder später mit XML spezifiziert. Erst recht für die Softwareevolution wird XML eine zentrale Rolle spielen. Die alten Programme und Datenbanken werden über XML-Schnittstellen in die neuen Systemarchitekturen eingebunden. XML-Schemen werden die Datenwelt festhalten. XML-Dokumente werden die unterschiedlichen Softwareschichten - Präsentations-, Steuerungs-, Verarbeitungs- und Zugriffsschicht - verbinden. Somit verspricht das Jahr 2002 das Jahr der Systemintegration mittels XML zu werden. In diesem Sinne wünsche ich allen „Softwerkern" ein erfolgreiches Jahr bei der Evolution und Integration Ihrer bestehenden Anwendungssysteme. Ich würde mich freuen, Sie auf dem einen oder anderem Seminar zu diesem Thema begrüßen zu dürfen.

Mit besten Grüßen aus Wiesbaden

Harry M. Sneed

Wiesbaden, im Januar 2002

UML - all the way down

So hieß der Gastvortrag von Ivar Jacobson auf der diesjährigen internationalen Software Maintenance Konferenz. Jacobson ist als Vertreter der Firma Rational gekommen, um zu proklamieren, daß UML die Programmsprache der Zukunft wird. Wir brauchen uns künftig nicht mehr mit C++ und Java, geschweige denn mit COBOL oder PL/I herumzuschlagen. Alles wird gleich in UML programmiert. Die neue Linie der Firma Rational heißt - all the way down with UML - bis zum Code.

Statt mit Codetexten sollen Programmierer mit Diagrammen arbeiten. Ist nicht ein Bild aussagekräftiger als tausend Wörter? Die Idee, Code aus Spezifikationen zu generieren ist nicht neu. Sie kommt immer wieder. Sie war das Leitmotiv der Programmgeneratoren à la Delta und SWT der 70er Jahre und das Ziel der CASE-Werkzeuge à la ADW, IEF und Excelerator in den 80er Jahren. Automatic Programming wurde am M. I. T. in den USA für fast 20 Jahre erforscht und schließlich aufgegeben. Es hat sich erwiesen, daß es viel schwieriger ist, eine vollständige formale Spezifikation zu verfassen, aus der verschiedene Programmversionen generiert werden können, als eine Programmversion direkt zu schreiben. Wenn aus der Spezifikation ein komplettes, lauffähiges Programm generiert werden soll, muß die Spezifikation auf der gleichen semantischen Ebene sein wie das Programm selbst. Also wird aus der Spezifikation nur eine weitere höhere Programmiersprache. Dies und noch mehr ist schon mehrfach dokumentiert worden.

Es sieht aber so aus, als ob die „drei Amigos von Rational" zu sehr mit dem Schreiben eigener Bücher beschäftigt sind, um die Bücher anderer Fachleute zu lesen. Von der Praxis sind sie schon länger entfernt, mit der Theorie des Software-Engineering haben sie auch nichts mehr im Sinn. Sie sind in einer eigenen virtuellen Welt abgehoben. Der Erfolg verblendet. Es ist nur eine Frage der Zeit, bis UML auch den Weg früherer Entwurfsmethoden und Spezifikationssprachen geht. Dann werden sich noch einige der heutigen OO-Anhänger an sie erinnern: „Einmal gab es diese drei Amigos und sie wollten Programme aus Diagrammen generieren. Was wird wohl aus denen geworden sein?"

Die Zweckentfremdung von CASE

CASE - Computer Aided Software Engineering - ist ursprünglich Anfang der 80er Jahre als Antwort auf die Notwendigkeit entstanden, den Softwareentwicklungsprozeß selbst mit Software zu unterstützen. Bis dahin wurden die Systementwurfsdiagramme - HIPO-, SADT- oder SA-Diagramme - manuell mit Hilfe von Plastikschablonen erstellt. Auch die Programmentwurfsdiagramme, wie Ablauf-, Balkendiagramme und Struktogramme, wurden von Hand gezeichnet, oft mit großer Akribie. Da dies alles sehr arbeitsintensiv war, stieg der Wunsch, Tools einzusetzen, um diese lästige Zeichenarbeit zu erleichtern.

Vorläufer der CASE-Bewegung waren die Professoren Teichrow in den USA, der das ISDOS-System entwickelte, und Lauber in Deutschland, der das EPOS-System entwickelte. Schon damals gab es die Unterscheidung zwischen Informations- und Prozeßsteuerungssystemen. ISDOS wurde konzipiert, um Informationssysteme zu modellieren, während EPOS für die Prozeßdatenverarbeitung gedacht war. Die Mutter aller CASE-Werkzeuge war aber das RXVP-System des US Defense Departments.

Dieser allumfassende CASE-Werkzeugkasten für die Entwicklung des ersten „Ballistic Missile Defense Systems" umfaßte drei Subsysteme mit jeweils drei verschiedenen Sprachen - RSL für die Spezifikation der Requirements, PDL für den Entwurf der Systemarchitektur und der Programmvorgaben und TSL für die Spezifikation der Testfälle in Form von Assertions oder Zusicherungen.

Von Integration war in RXVP keine Rede, im Gegenteil, alle drei Sichten der Software sind unabhängig voneinander entstanden - die Requirementspezifikation in Alabama, die Programmentwürfe in Texas und die Testfälle in Kalifornien. Die Programme selbst sind in FORTRAN von einer vierten Gruppe in Los Alamos geschrieben worden, und zwar in Anlehnung an die aus Alabama stammende Requirementspezifikation und die aus Texas stammenden Programmentwürfe. Damals legte man großen Wert darauf, die Sichten auf das System auseinander zu halten, um durch den Abgleich der unterschiedlichen Sichten Inkonsistenzen und Unvollständigkeiten besser aufdecken zu können. Im Vordergrund standen die Zuverlässigkeit und Sicherheit. Zeit und Kosten waren sekundäre Ziele.

In den 80er Jahren entstanden die ersten kommerziellen CASE-Tools, um die Funktions- und Datenmodellierung zu unterstützen. Die ersten Werkzeuge waren auf eine bestimmte Diagrammart ausgerichtet, wie z. B. der E/R-Modellierer von Peter Chen. Später haben die CASE-Tools immer mehr Diagrammarten - wie Baum-, Datenfluß- und E/R-Diagramme - angeboten. Bis Mitte der 80er Jahre waren alle gängigen Entwurfsmethoden, wie Structured Analysis und Structured Design, mit CASE-Tools abbildbar. Dieser Autor hat selbst ein solches Werkzeug konzipiert und in Ungarn entwickeln lassen. Darin wurden Balken- mit Flußdiagrammen, E/R-Diagramme und Entscheidungstabellen kombiniert. Dazu gab es Texte und eine eigene Testsprache für die Verfassung von Testfällen. Die Dualität von Funktions- und Testspezifikation wurde darin bewahrt. Vom SofSpec aus war es möglich, Testprozeduren

automatisch zu generieren. Die Spezifikation blieb als Basis für den Test der getrennt erzeugten Programme sowie im RXVP das große Vorbild.

Der wichtigste Kern aller CASE-Werkzeuge ist schon seit jeher das Repository - in der Regel eine vernetzte oder relationale Datenbank. Hier werden die Entitäten und Beziehungen der diversen Diagrammarten abgespeichert. Damit entsteht so etwas wie eine Projektdatenbank, ein Behälter für alle Softwareelemente, Beziehungen und Strukturen des geplanten Anwendungssystems. Das Repository bietet nicht nur die Möglichkeit, Diagramme und Berichte zu generieren, sondern auch Abfragen bezüglich der Systemzusammenhänge zu beantworten.

In der zweiten Hälfte der 80er Jahre kam die Anforderung auf, aus dem CASE-Repository auch noch Programme zu generieren. Wieso brauchte man noch Programmierer, um Code zu schreiben, wenn alle Informationen zur Erstellung der Programme elektronisch gespeichert vorlagen? Man konnte den Computer benutzen, um sich selbst zu programmieren. Es erschienen Bücher von James Martin wie „Programming without Programmers" und „CASE = Software-Automation". Die Manager waren natürlich begeistert - in Zukunft sollten die Anwender in der Lage sein, ihre Systeme selber zu entwickeln. Auch der Autor mußte dem Druck des allmächtigen Marktes nachgeben und vom SofSpec aus über SoftCon und SoftGen Batch- und CICS-Online-Programme generieren.

So kam es, daß die Prinzipien der Mutter aller CASE-Werkzeuge - RXVP -, wo Spezifikation und Implementation stets getrennt waren, über Bord geworfen wurden. In den neuen CASE-Werkzeugen sind Vorgabe und Realisierung eins geworden. Dadurch ist die Spezifikation bzw. Vorgabe immer komplexer geworden, so daß es am Ende kaum noch von Programmierern beherrschbar war, geschweige denn von Analytikern und überhaupt nicht von Anwendern. Die Programmierer zogen es vor, wieder Maschinensprache in C bzw. C++ zu codieren, die Analytiker zogen es vor, wieder Prosatexte ergänzt durch einige Freistildiagramme, zu verfassen und die Anwender kehrten zu Prototypen zurück.

Es hat sich bestätigt, was die Forscher Rich und Water von M. I. T. nach jahrelanger Beschäftigung mit dem „Automatic Programming"-Projekt festgestellt haben, nämlich es sei viel schwieriger und aufwendiger, eine vollständige, widerspruchsfreie Spezifikation eines Programmes zu verfassen als das Programm selbst.

Leider sind alle OO-CASE-Tool-Anbieter, Rational voran, im Begriff, den gleichen Fehlweg einzuschlagen. Nach dem Motto „UML - all the way down" kann man nur müde lächeln. „Don't worry, tomorrow everything will be a lot better". Diesen Satz haben wir irgendwann schon mal gehört.

Warum Spezifikation und Programmierung getrennt bleiben müssen

Die Idee, aus einer Spezifikation Code zu generieren, ist sehr verheißungsvoll und kommt immer wieder. Viele fragen, warum soll eine Lösung zweimal beschrieben werden, wenn einmal genügt. Dabei übersehen die Protagonisten dieser Idee etwas Grundsätzliches. Die Korrektheit bzw. Wahrheit einer Lösung läßt sich nur bestätigen, wenn sie mit einer anderen Lösungsbeschreibung abgeglichen wird. Das ist der Sinn von Verifikation. Etwas ist wahr, wenn es der Wahrheit entspricht. Diese Wahrheit muß jedoch dokumentiert sein.

Das haben die Väter von Software-Engineering gewußt, und das haben die Erfinder des ersten CASE-Tools - RXVP - praktiziert. Die Spezifikation soll dazu dienen, mit dem Programm verglichen zu werden, und zwar sowohl statisch als auch dynamisch. Die Spezifikation ist eine Beschreibung, wie ein Problem zu lösen sei, aus der Sicht eines Fachspezialisten. Der Programmcode ist wiederum die Beschreibung, wie ein Problem gelöst wurde, aus der Sicht eines Realisierers. Es gilt, die beiden Sichten gegeneinander abzugleichen, um Abweichungen erkennen zu können. Dies ist aber nicht möglich, wenn beide Sichten aus einem Guß stammen. Sie müssen aus unterschiedlichen Quellen hervorgehen. Allerdings müssen sie einander zuzuordnen sein, entweder über gleiche Begriffe oder über Querverweise.

Aus der Spezifikation werden die Funktionen, Datenstrukturen, Ablauffolgen, Bedingungen und Verarbeitungsregeln abgeleitet, um mit den im Code realisierten Funktionen, Datenstrukturen, Abläufen, Bedingungen und Algorithmen verglichen zu werden. Dies kann manuell durch Reviews und Inspektionen oder maschinell über Repositoryverknüpfungen geschehen.

Aus der Spezifikation werden auch die Testfälle abgeleitet, um die Programme zu testen. Diese Fälle beschreiben die Zustände vor und nach der Testausführung. Wenn beide erfüllt sind, gilt der Test als bestanden. Wenn Spezifikation und Programm identisch sind, hat dieser Abgleich keinen Sinn mehr.

In dem Test der ersten „Ballistic Missile Defense Software" in den 70er Jahren stellte sich heraus, daß ca. 40 % der Abweichungen auf Fehler in der Requirementspezifikation zurückzuführen waren. Daß die Spezifikation nicht wesentlich fehlerfreier sein kann als die Implementation, war zu erwarten. Wichtig war, daß die Unterschiede in den Sichten überhaupt zum Vorschein kamen. Dies war jedoch nur über den Abgleich zweier unterschiedlicher Sichten möglich.

Da der Beweis der Korrektheit und der Einhaltung der Produktziele, ob Zuverlässigkeit, Sicherheit oder Performance, nach wie vor das Hauptproblem der Softwareentwicklung ist, müssen Spezifikation und Realisierung weiterhin getrennt bleiben.