**Die Unified Modeling Language  
im Kontext des Projektmanagements von Software-Systemen**

Gliederung

1. Einleitung (3 Seite)
   1. Grundlagen des Projektmanagements
      1. Definition
      2. Modelle
      3. Phasen
      4. Prozesse
2. UML (9 Seiten)
   * 1. Definition
     2. Historie
     3. Struktur
     4. Anwendung
     5. Abstraktionsgrad
     6. Strukturdiagramme
        1. Verteilungsdiagramm
        2. Komponentendiagramm
        3. Paketdiagramm
        4. Klassendiagramm
     7. Dynamische Diagramme
        1. Anwendungsfalldiagramm
        2. Aktivitätsdiagramm
        3. Zustandsdiagramm
     8. Werkzeuge
3. Zusammenfassung (3 Seite)
   1. Ausblick
   2. Fazit
   3. Bewertung (Vorteile/Nachteile)
4. Einleitung

Die Unified Modeling Language (UML) ist einer der wichtigsten Methoden, um Software-Systeme in Modellen zu beschreiben. Diese Arbeit soll in die UML einführen, ein Überblick über die wichtigsten Elemente geben und einen Eindruck vermitteln, welches die Stärken und Schwächen bei der Softwaremodellierung im Kontext des Projektmanagement darstellen.

Zunächst werden in diesem Kapitel die Grundlagen des Projektmanagements in Bezug auf Softwareprojekte vorgestellt. Insbesondere das Verständnis der einzelnen Phasen der Softwareentwicklung und der verschiedenen Prozessmodelle im Bereich der Analyse und des Designs sind wichtig. Die UML zielt auf die Spezifikation des Systems und dem darauf folgenden Entwurf der Architektur ab.

In Kapitel zwei wird zunächst die Definition der UML und deren Historie beschrieben. Darauf folgt ein Überblick über die Struktur, Anwendungsbeispiele und den wichtigsten Diagrammen der UML.

Im abschließenden dritten Kapitel wird das Wichtigste zusammengefasst. Dazu erfolgt eine Bewertung über die Vor- und Nachteile der UML im Einsatz mit verschiedenen Prozessmodellen, die das Projektmanagement bietet.  
Ein Ausblick schließt die Arbeit ab.

* 1. Grundlagen des Projektmanagements
     1. Definition

In der Lehre wird unter dem Begriff Projektmanagement folgende Definition verstanden: „Projektmanagement ist ein Leitungs- und Führungskonzept für Projekte, welches den Entwicklungsprozess definiert, die notwendigen Aufgaben definiert, die Methoden für die Lösung der Aufgaben vorschlägt, Institutionen schafft und nutzt, von denen diese Aufgaben realisiert werden können und abteilungsübergreifend arbeitet.“{1} Diese Definition ist unabhängig von der Branche, in dem das Projekt umgesetzt werdem soll. Diese Arbeit beschränkt sich jedoch auf Projekte, bei denen das Ziel ein Software-System ist. Um die Aufgaben, die für die Umsetzung nötig sind, definieren, Methoden festlegen und danach mit der Realisierung beginnen zukönnen, müssen die Anforderungen an das Software-System zunächst festgestellt, dokumentiert und kommuniziert werden. Dazu werden Modelle benötigt. Diese Modelle dienen als Abstraktion des angestrebten Software-Systems, als Abstraktion des zeitlichen Ablaufes des Projektmanagements im Großem und als Abstraktion der einzelnen Teile des Entwicklungsprozesses und deren Methoden im Kleinen.

* + 1. Modelle

Im Kontext von Software-Systemen werden Modelle erstellt, um existierende oder zukünftige Systeme besser zu verstehen. Ein Modell entspricht jedoch nie genau der Realität. Modellieren bedeutet immer hervorheben und weglassen. Hervorheben des Wesentlichen und weglassen unwichtiger Details. Was aber ist wesentlich, was unwichtig? Diese Frage kann nicht allgemein beantwortet werden. Vielmehr ist die Antwort immer abhängig davon, was mit dem Modell für Ziele verfolgt werden und wer die Betrachter oder Leser des Modells sind.

Je mehr Informationen in einem Modell gezeigt werden sollen, desto komplexer und schwieriger wird es. Eine Landkarte von Europa zum Beispiel, die gleichzeitig politische, geologische, verkehrstechnische und demographische Informationen enthält, ist kaum mehr lesbar. Die Lösung des Problems liegt darin, die verschiedenen Informationen auf einzelne Landkarten zu verteilen. Es werden verschiedene Sichten auf den betrachteten Gegenstand gebildet. Diese Sichten sind miteinander in vielfältiger Weise verbunden. Wenn eine Sicht geändert wird, müssen in der Regel alle anderen Sichten ebenfalls angepasst werden. Wenn beispielsweise in den Niederlanden der Nordsee neues Land abgetrotzt wird, müssen alle Sichten, d.h. alle Landkarten, angepasst werden.

Das Modell eines Systems muss in der Regel folgende Aufgaben erfüllen: Ermöglichen der Kommunikation zwischen allen Beteiligten.

Um das richtige System zu bauen, ist es unerlässlich, dass alle Beteiligten ein gemeinsames Verständnis des Systems haben. Die verwendete Terminologie muss von allen verstanden werden, die Auftraggeber müssen sich auf gemeinsame Anforderungen einigen, die Entwickler müssen diese Anforderungen verstehen können, gefällte Entscheidungen müssen auch nach Monaten noch nachvollzogen werden können. Bei all diesen Punkten geht es letztlich um Kommunikation zwischen den Beteiligten über einen gewissen Zeitraum hinweg. Dies ist ohne Aufzeichnungen, ohne Repräsentation der gefundenen Fakten nicht möglich.

Visualisierung der Fakten für die Auftraggeber, Fachexperten und

Anwender

Die Fakten, die zu einem System gesammelt werden, müssen so repräsentiert werden, dass sie von den Betroffenen verstanden werden. Dies ist oft einfacher durch ein grafisches Modell als durch rein textliche Beschreibungen zu erreichen. Nach unserer Erfahrung stößt man jedoch in der Praxis häufig auf reflexartigen Widerstand, will man mit Diagrammen statt mit Texten kommunizieren. Es ist notwendig, diese Widerstände zu überwinden. Oft steckt die Angst vor Neuem dahinter und die Diagramme sehen auf den ersten Blick etwas kompliziert aus. Deshalb enthält dieses Buch für jedes Diagramm eine Anleitung, wie dieses gelesen werden muss.

Überprüfung der Fakten bezüglich Vollständigkeit, Widerspruchsfreiheit und Korrektheit

Ein (mehr oder weniger) formales Modell ermöglicht es, die gefundenen Fakten bezüglich Vollständigkeit, Widerspruchsfreiheit und Korrektheit zu überprüfen. Durch die klare Darstellung insbesondere von Zusammenhängen, wird es möglich, gezielt Fragen zu stellen und zu beantworten. Wir werden diese Fragen bei den einzelnen Diagrammen jeweils auflisten.

* + 1. Projektphasen

Von der Initiierung eines Projekts bis zu seinem erfolgreichen Abschluss durchläuft es verschiedene Projektphasen. Die Phasen unterteilen den zeitlichen Ablauf für das Projektmanagement. Im Allgemeinen wird mit der Ausarbeitung der Projektziele und -inhalte begonnen und daran anschließend muss die Planung von Arbeitspaketen mit Terminen, Kosten u. a. vorgenommen werden. Nach dieser Planungsphase sind die Aufgaben zu kontrollieren und zu steuern sowie je nach Notwendigkeit Plananpassungen vorzunehmen. Ob erfolgreich oder nicht erfolgreich – eine Auswertung zum Abschluss eines Projekts sollte die Ursachen aufzeigen und den Projektablauf dokumentieren. Die folgenden Phasen können somit spezifiziert werden:

Start -> Planung -> Kontrolle -> Abschluss

Aufgaben der Systemanalyse {6}:

* + - * + Festlegen der Grenzen und der Schnittstellen zur Umwelt des Systems
        + Festlegen der Funktions-Anforderungen, die das System erfüllen soll
        + Modellierung des Begriffsmodells
    1. Prozessmodelle

1. UML
   1. Definition

Die Unified Modeling Language (UML) ist eine Menge von Notationselementen, mit denen Modelle für Softwaresysteme entwickelt werden können. Dies betrifft die Analyse, das Design und ganz allgemein die Darstellung und Dokumentation der Softwareelemente oder des Softwareverhaltens.

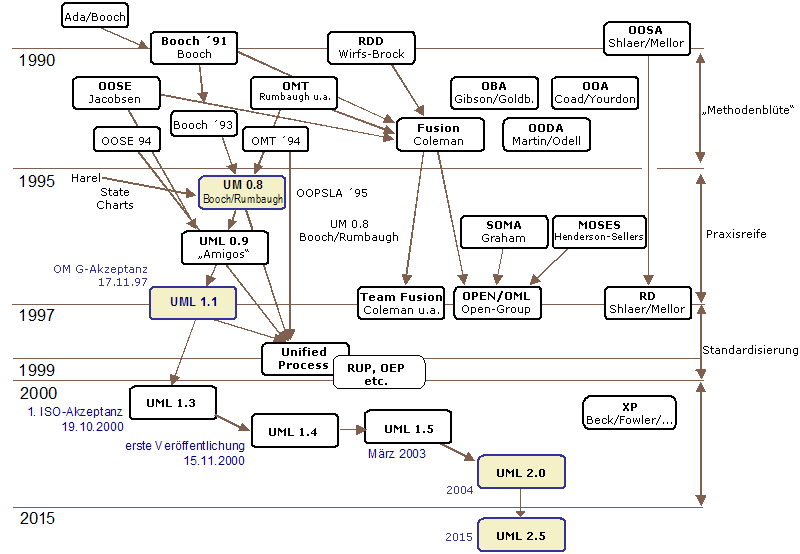
* 1. Historie

Die Beschäftigung mit der Thematik der objektorientierten Analyse / Design kam erst in den 90er Jahren in Schwung. Autoren, die sich hier früh einen Namen gemacht haben waren Adele Goldberg, Grady Booch, Peter Coad, Edward Yourdon, James Rumbaugh, Bertrand Meyer und Ivar Jacobsen.

Schon früh setzten sich die Methoden von James Rumbaugh und Grady Booch durch. Beide konkurrierten jedoch. Rumbaughs Methode schien besser für die Analyse zu sein und Boochs Methode war wohl beim Design vorteilhafter. James Rumbaugh wurde zudem 1994 von der Rational Software Cooperation eingestellt, und war dort auch für die Weiterentwicklung der IBM Toolsuite verantwortlich. Mit den Kräften von Ivar Jacobsen vereint, versuchten Rumbaugh und Booch dann ihre Differenzen zu überbrücken und eine einheitliche Sprache zu erschaffen. Dies natürlich auch etwas auf Druck der Industrie hin. Denn schließlich ließ sich mit entsprechenden Werkzeugen und Vorgehensmodellen auf Basis von UML durchaus Geld verdienen.

Ivar Jacobsens Firma Objectory wurde dann auch von Rational gekauft und aus der Vereinheitlichung der Methoden entstand die Unified Method (UM). Booch, Rumbaugh und Jacobsen wurden fortan die drei Amigos genannt.

Der Druck wurde jedoch größer, eine nicht proprietäre Sprache auf Basis der UM zu entwickeln: Die Unified Modeling Language (UML). Ein internationales Konsortium entwickelte 1996 die UML und übergab sie dann an die OMG. So entstand die UML 1.1 im Jahre 1997. Es entwickelte sich dann in Schritten zur Version 2.0, die bis 2006 die offiziell letzte downloadbare Version war. Momentan ist die Version 2.5 vom Juni 2015 im Downloadbereich der OMG verfügbar: <http://www.omg.org/spec/UML/2.5/PDF/>

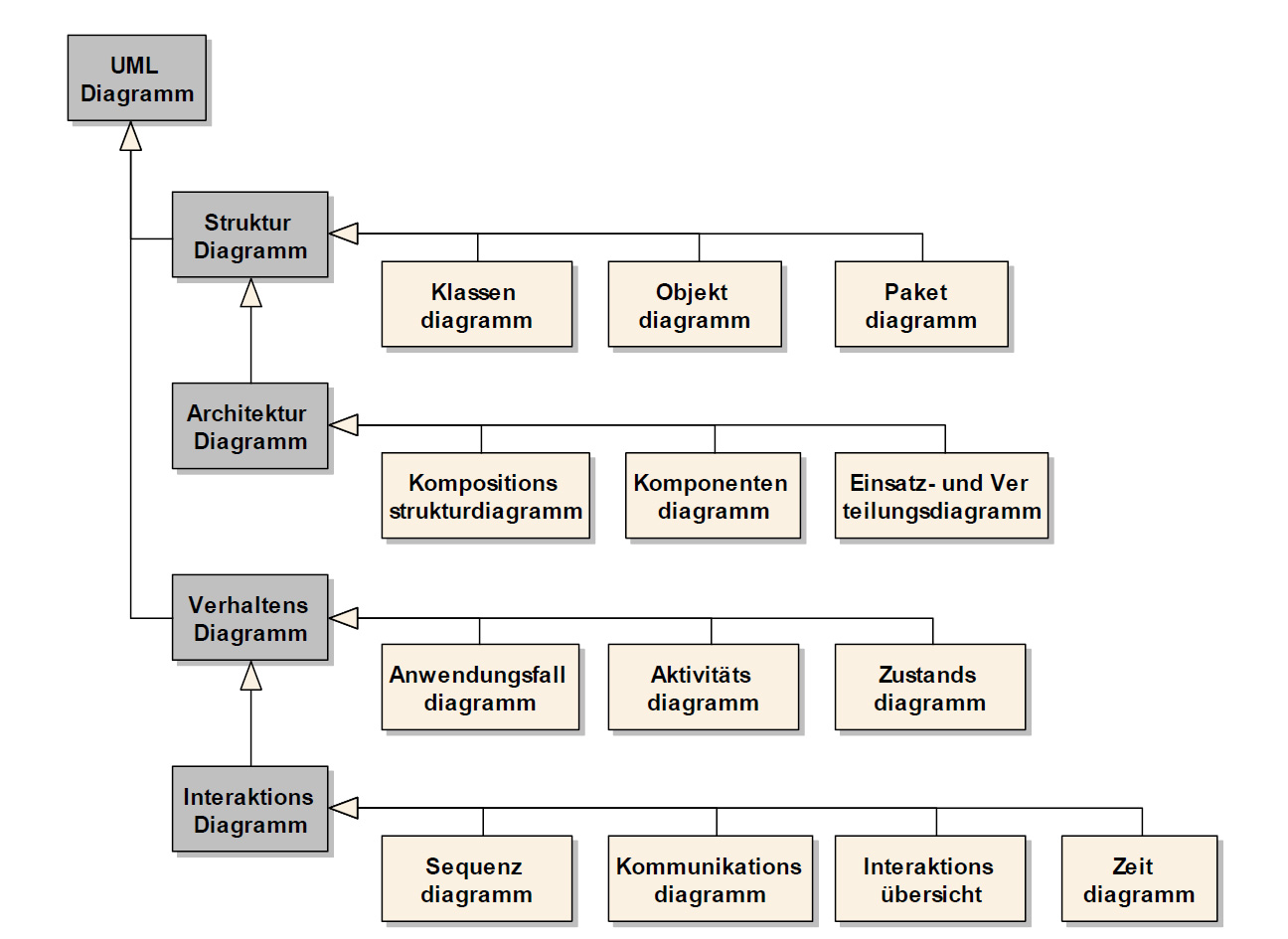


* 1. Struktur

Modell – Sichten - Diagramme

Strukturdiagramme und Architekturdiagramme werden auch als statische Diagramme bezeichnet. Verhaltensdiagramme und Interaktionsdiagramme werden oft auch als dynamische Diagramme bezeichnet.

Siehe Rumpe



* 1. Anwendung

Der eigentliche Sinn der UML liegt darin, Ideen zu kommunizieren!

Die Frage ist immer, wie kann man am besten und schnellsten das Design eines Systems herüberbringen und es dokumentieren. Es ist klar, dass UML vielleicht nicht immer das beste Mittel sein muss und auch nicht vollständig sein kann.

Eine - bisher wenig dokumentierte - Anwendungsmöglichkeit der UML ist das **Reverse-Engineering**. Hierbei haben Sie vielleicht schon ein System, das weiterentwickelt werden soll und Sie möchten sich einen Überblick über das System verschaffen. Gute Werkzeuge können den bestehenden Code analysieren und daraus verschiedene UML-Diagramme generieren. Es kann sehr hilfreich sein, sich aus einem Berg von Code erst einmal einige Diagramme (z. B. Klassen-, Package- oder Komponentendiagramme) zeigen zu lassen und so das Programm erst einmal meta zu verstehen.

Reverse Engineering ist ein nicht zu unterschätzender Ansatz in der Industrie, wenn es darum geht Wettbewerbsvorteile zu erreichen. Ein bekanntes Beispiel für Reverse Engineering ist die Tabellenkalkulation Lotus 1-2-3, das Programm wurde von Microsoft zerlegt, verstanden und als MS Excel neu gebaut. Auch das ist Reverse Engineering.

Neben der schriftlichen Dokumentation einer Architektur von Architekten wird UML oft mehr informell in Teamdiskussionen quasi als Sketch verwendet. Im Sketch Mode wird informelles UML verwendet und die dargestellten UML Diagramme zeigen meist nur die interessanten Ausschnitte des Systems, das gerade besprochen werden soll.

**(sketch (engl.) = Skizze)**

Im erstgenannten Blueprint-Mode ist meistens der Architekt des Systems damit beschäftigt, das Design der Software mehr oder weniger vollständig zu hinterlegen (blueprint = Blaupause; ein Begriff aus der Architektur).

Sketch- und Blueprint-Mode sind nicht synonym zu verwenden. Sketch ist eine kleine Diskussion über Ausschnitte. Blueprint ist eine größere Architekturvorgabe.

**Programmiersprache / MDA**

Ein wichtiges Element der UML ist, dass es selbst von Maschinen weiterverarbeitet werden kann. Die UML und die dazugehörigen Sprachen wie XMI (XML Metadata Interchange) werden von Frameworks gelesen, die dann auch Code generieren können.

Das Entwickeln von Software ist relativ teuer, gerade europäische IT-Firmen müssen sehen, wie Sie konkurrenzfähig und schnell, qualitativ (!) hochwertige Software erstellen können.

Dies kann sicher teilweise dadurch geschehen, dass Code automatisiert generiert wird. Wenn UML Diagramme die Komponenten, Pakete, Klassen und auch noch das Verhalten von Software beschreiben, wieso soll man dann nicht möglichst viel Code generieren lassen? In diesem Fall wäre dann die UML selbst so etwas wie eine höherwertige Meta-Programmiersprache.

Dieses Vorgehen führt uns zum Forschungsgebiet der MDA (Model Driven Architecture), wofür es bereits erstaunlich viele leistungsfähige Frameworks gibt. Leider sind viele dieser Toolkits in der Industrie versteckt. Es gibt aber auch einige gute freie Toolkits z. B. unter icoWeb AndroMDA .

* 1. Abstraktionsgrad

Im Kapitel 1.1.3 wurde die Aufgabe von Modellen dargestellt. Um die Vielzahl der UML-Diagramme besser einordnen zu können und eine Reihenfolge für die Erstellung vorzuschlagen, kann man sich am Abstraktionsgrad der Diagramme orientieren. Wie in Abbildung XX zu sehen ist, können fünf Ebenen unterschieden werden. Der Abstraktionsgrad nimmt von eins bis fünf immer weiter ab. Dass heißt, dass in der ersten Ebene die Vereinfachung und damit die Entfernung von dem Software-System, zu dem die Diagramme gehören, am größten ist. Ohne zu wissen, wie auf Ebene fünf das System funktionieren wird, sollte sich bereits zum Projektstart und in der frühen Phase der Projektplanung für die Analyse mit den Stakeholdern in der ersten Abstraktionsebene mit Hilfe des Anwendungsfall- und Aktivitäts-Diagramm ausgetauscht werden.

Jede Ebene zeigt in Abhängigkeit mit dem jeweiligen Abstraktionsgrad eine andere Sicht des Systems und verfolgt damit andere Ziele. Erst wenn die Anforderungen an das System festgelegt wurden, macht es Sinn in der zweiten Ebene das Verteilungs-Diagramm und das Zustands-Diagramm zu erstellen. Danach folgt in Ebene drei das Paket-Diagramm, in Ebene vier das Komponenten-Diagramm und in Ebene fünf das Klassen-Diagramm. Wird später etwas in einer höheren Ebene geändert, so muss in allen Ebenen darunter geprüft werden, ob dies auch dort eine Änderung zur Folge hat. Anders herum ist dies nicht nötig. Hält man sich beim Modellieren der UML-Diagramme an diese Reihenfolge der Abtraktionsgrade, dann spart man viel Arbeit durch nachträgliche Änderungen.

1. Anwendungsfalldiagramm (Use-Case) und Aktivitäts-Diagramm

2. Verteilungs-Diagramm und Zustandsdiagramm

3. Komponenten-Diagramm

4. Paket-Diagramm

5. Klassen-Diagramm

* 1. Strukturdiagramme
     1. Verteilungsdiagramm

Das Verteilungsdiagramm (engl. Deployment Diagram) wird relativ früh spezifiziert. Relativ bald nach den Use-Cases. In diesem Diagramm geht es darum, die physische Struktur der gesamten Anwendung zu zeigen. Also welche Komponenten an welchem Ort auf welchen physischen Geräten laufen.

Dies ist für Studenten oft schwer nachzuvollziehen, da man in der Regel nur kleine Projekte schreibt, die nur auf einem Rechner laufen.

Beispielsweise gibt es im Projekt TollCollect verschiedene Rechenzentren, Satelliten, Geräte auf Autobahnen und Geräte in den Raststätten. Es läuft also ganz unterschiedliche Software auf sehr verschiedenen Rechnern an vielen Stellen.

Das Ziel des Verteilungsdiagrammes ist es, die physische Verteilung schlüssig darzustellen.

* + 1. Komponentendiagramm

Das Komponentendiagramm (engl. Component Diagram) beschreibt eine bestimmte Menge von Komponenten. Das Komponentendiagramm ist die Antwort auf die Frage, wie das konkrete System strukturiert ist.

In der Regel sind es mehrere Klassen, es kann aber auch nur eine Klasse sein, die aber keinesfalls Details zeigt. Schon im Gehe zuDeployment-Diagramm war eine Menge von Komponenten zu sehen, die einzeln oder zusammen eine oder mehrere bestimmte Aufgaben erfüllen.

Wichtig dabei ist, dass man Abhängigkeiten der Komponenten spezifizieren kann und auch sollte. Komponenten bieten Schnittstellen, die von anderen Komponenten genutzt werden können.

BERND ÖSTERREICH schreibt dazu: Eine Komponente ist ähnlich wie eine Klasse instanziierbar und kapselt komplexes Verhalten. Mit ihr werden Einheiten gebildet, die eine hohe fachliche Kohärenz haben. Im Gegensatz zu einer Klasse wird bei einer Komponente zusätzlich auch die prinzipielle Austauschbarkeit (Substituierbarkeit) angestrebt. Eine technische Plattform für Komponenten ist beispielsweise Enterprise Java Beans (EJB).

* + 1. Paketdiagramm

Das Paketdiagramm strukturiert den Namensraum von Paketen, Komponenten oder Klassen.

Es korrespondiert üblicherweise mit den Packages in Java oder mit den Namespaces in C#. Beispielsweise de.vfh.portal.datenbank in der die Klasse Zugriffsschicht enthalten sein kann. Wichtig dabei ist eine hierarchische Struktur, welche die eindeutige Identifizierung des Modellelementes ermöglicht.

Warum sind Paketdiagramme so wichtig?

In der Praxis sind Paketdiagramme wichtig, weil die Entwickler bereits frühzeitig nach einer Spielwiese fragen und schon Testcode entwickeln wollen.

Pakete können wie Komponentendiagramme, die wirkliche Architektur des Systems wiederspiegeln. Design Patterns und Schichten in der Anwendung werden hier transparent. Wichtig ist, dass die Transparenz frühzeitig gegeben ist, von vornerein ausreichend darüber diskutiert wird oder erfahrene Designer zum Einsatz kommen.

Am wichtigsten sind Paketdiagramme, weil man mit ihnen die Abhängigkeiten modellieren kann. Versuchen Sie also das Paket zu zeichnen und sich dann zu fragen: Wer benutzt wen? Zeichnen Sie das mit Pfeilen in das Diagramm hinein. Dies zeigt Ihnen die Benutzt-Struktur. Diese Benutzt-Struktur ist für ein gutes Design bzw. eine gute Architektur extrem wichtig.

Liegt hier eine völlig wirre Struktur vor, wer wen benutzt? Oder gibt es einen klaren Fluss von oben nach unten? Letzteres ist für eine änderbare Architektur sehr wichtig.

Beispielsweise kann man in einem Paketdiagramm, welches benutzte Beziehungen enthält, auch sehr schön erkennen, ob es Zyklen in der Verwendung gibt. Also verwendet Paket A das Paket B und umgekehrt? Wenn dies der Fall sein sollte, ist oftmals ein Redesign vonnöten.

Im letzten Kapitel sehen wir, dass es viele Werkzeuge gibt, die die Benutztrelation zwischen Paketen automatisch analysieren und auf Probleme hinweisen.

Aber erleben Sie das jetzt schon einmal selbst, indem sie ein Paketdiagramm so gestalten, dass neben dem hierarchischen Namensraum auch eine hierarchische Benutztrelation vorliegt. Ihr Code wird wartbarer und unempfindlicher bei Änderungen.

* + 1. Klassendiagramm

Ein Klassendiagramm visualisiert Klassen auf verschiedene Art und Weise. Dazu werden einmal die Klassen selbst dargestellt, ggf. auch deren Beziehungen zueinander und evtl. auch deren innnere Struktur. Üblich sind zwei Ziele:

Darstellung der Klassen und Beziehungen: Hierbei werden verschiedene Klassen eines Paketes oder einer Komponente im Diagramm dargestellt. Dabei ist es wichtig zu zeigen, welche Klassen es überhaupt gibt und welche Klasse von anderen verwendet wird. Letzteres nennt man Assoziationen.

Darstellung der inneren Struktur: Der Autor des Klassendiagrammes will dann zeigen oder definieren, welche Attribute oder Methoden die Klassen haben.

Beide Formen werden oft miteinander kombiniert und bilden dann ein ausführliches Klassendiagramm.

* 1. Dynamische Diagramme
     1. Anwendungsfalldiagramm
     2. Aktivitätsdiagramm
     3. Zustandsdiagramm
  2. Werkzeuge

Damit die Diagramme nicht mit dem Stift zu Papier gebracht werden müssen, gibt es eine Vielzahl von Tools, die die mächtige UML-Notation beherrschen. Neben Online-Angeboten, stehen Desktop-Versionen zum Download bereit. Doch nur wenige sind kostenlos und unterscheiden sich sehr stark in ihren Fähigkeiten. Einige Tools arbeiten mit der Entwicklungsumgebung zusammen, mit der man das Software-System programmiert. Andere können aus den modellierten UML-Klassen fertigen Code für einen Prototypen generieren. Eine umfassende Liste {10} stellt die Firma OOSE im Internet bereit.

Wichtig zu beachten ist, dass derzeit kein Austausch der Diagramme zwischen den einzelnen Tools verschiedener Tool-Hersteller möglich ist. Dies könnte sich aber mit der neuen UML-Spezifikation 2.5 und dem Kapitel über Diagrammaustausch ändern.

1. Zusammenfassung
   1. Fazit
   2. Bewertung (Vorteile/Nachteile)

Die Bedeutung von UML ist in der Realität noch viel umstrittener, als es den Anschein hat.

Im akademischen Bereich wird UML an vielen Stellen gelehrt, da es noch viele Befürworter auf Seiten der Hochschule gibt. Nicht selten wird dies dann auch in der Industrie unkritisch übernommen.

Dennoch gibt es Unternehmen und Bewegungen die die Nützlichkeit von UML viel geringer bewerten. Gerade auch im agilen Umfeld wird deshalb oft auf andere Techniken gesetzt. Dies ist bereits aus dem mittlerweile recht alten XP Modell zu sehen, welches auch keine umfangreichen UML Artefakte vorsieht. In einigen bedeutenden Unternehmen wie bspw. Google ist UML sogar verboten!

Das soll nicht heißen, dass UML vollständig untersagt ist. Wenn zwei Entwickler sich unbedingt mit UML unterhalten möchten, so ist dies kein Problem. Jedoch wird die Kraft eines Prototypen oder eine prototypische Architektur in der Unternehmenskultur als viel effizienter angesehen als ein UML-Papier.

Entscheidend für jeden Softwaretechniker ist daher, das Spektrum der Werkzeuge oder Kommunikationsmittel zu kennen und selbst beurteilen zu können, inwieweit UML ein geeignetes Mittel ist oder ob andere Verfahren besser geeignet sind.

Vorteile

Das beschriebene Modell in der UML spiegelt das reale System wider. Es besteht die Möglichkeit, Fehler im Modell früh und durchgehend zu beheben. Zwischenschritte wie etwa die Ergänzung des Codes außerhalb des Modell-Designs entfallen. Es ist die Sprache, die sich in der Industrie durchgesetzt hat und welche Sie auf jeden Fall beherrschen, dennoch kritisch betrachten sollten. Es besteht die Möglichkeit, das gleiche Modell auf verschiedenen Plattformen (Hardware wie auch Software) lauffähig zu machen. Ein weiterer wichtiger Nutzen der UML liegt darin, Details zu verbergen und später dann diese mit oder ohne UML verfeinern zu können. In vielen Fällen müssen Sie sich dem System Top-Down nähern und aus der Vision / Analyse konkrete Teile oder Komponenten spezifizieren. Kommunikation mit Perseonen, die keine IT-Kenntnisse haben, aber UML verstehen können!

Nachteile

Als Preis für diese Vorteile muss eine tief greifende und exakte Kenntnis der UML erarbeitet und ein erheblicher Aufwand in die Erstellung der Modelle investiert werden.

* 1. Ausblick

Der nächste Schritt wäre, dass die Toolhersteller sich an der neuen Spezifikation orientieren und einen gemeinsamen Standard für den UML-Diagramm-Austausch schaffen.

Anmerkungen und To-do:

* + - * + Keine Notation, sondern Verständis für die Einordnung, Verwendung im Projektmanagement
        + Abgabe am 11.01.2016

Definitionen:

Projektmanagement ist ein Leitungs- und Führungskonzept für Projekte, welches den Entwicklungsprozess definiert, die notwendigen Aufgaben definiert, die Methoden für die Lösung der Aufgaben vorschlägt, Institutionen schafft und nutzt, von denen diese Aufgaben realisiert werden können und abteilungsübergreifend arbeitet. (Skript)

Die Unified Modeling Language (vereinheitlichte Modellierungssprache), kurz UML, ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen.[2] Sie wird von der Object Management Group (OMG) entwickelt und ist sowohl von ihr als auch von der ISO (ISO/IEC 19505 für Version 2.4.1[3]) standardisiert. Im Sinne einer Sprache definiert UML dabei Bezeichner für die meisten bei einer Modellierung wichtigen Begriffe und legt mögliche Beziehungen zwischen diesen Begriffen fest. UML definiert weiter grafische Notationen für diese Begriffe und für Modelle statischer Strukturen und dynamischer Abläufe, die man mit diesen Begriffen formulieren kann.

UML ist heute die dominierende Sprache für die Softwaresystem-Modellierung. Der erste Kontakt zu UML besteht häufig darin, dass Diagramme in UML im Rahmen von Softwareprojekten zu erstellen, zu verstehen oder zu beurteilen sind:

Projektauftraggeber und Fachvertreter prüfen und bestätigen zum Beispiel Anforderungen an ein System, die Wirtschaftsanalytiker bzw. Business Analysten in Anwendungsfalldiagrammen in UML festgehalten haben;

Softwareentwickler realisieren Arbeitsabläufe, die Wirtschaftsanalytiker bzw. Business Analysten in Zusammenarbeit mit Fachvertretern in Aktivitätsdiagrammen beschrieben haben;

Systemingenieure installieren und betreiben Softwaresysteme basierend auf einem Installationsplan, der als Verteilungsdiagramm vorliegt.

Die grafische Notation ist jedoch nur ein Aspekt, der durch UML geregelt wird. UML legt in erster Linie fest, mit welchen Begriffen und welchen Beziehungen zwischen diesen Begriffen sogenannte Modelle spezifiziert werden – Diagramme in UML zeigen nur eine graphische Sicht auf Ausschnitte dieser Modelle. UML schlägt weiter ein Format vor, in dem Modelle und Diagramme zwischen Werkzeugen ausgetauscht werden können. (wikipedia)

Literaturverzeichnis:

* + - * 1. Skript
        2. Matthias Geihros: IT-Projektmanagement
        3. Bernd Oestereich, Axel Scheithauer: Die UML-Kurzreferenz 2.5 für die Praxis
        4. Ernst Tiemeyer: Handbuch IT-Projektmanagement
        5. Bernhard Rumpe: Modellierung mit UML
        6. Chris Rupp, Stefan Queins: UML Glasklar
        7. Patrick Grässle, Henriette Baumann, Philippe Baumann: UML 2 projektorientiert
        8. Dietmar Steinpichler, Horst Kargl: Projektabwicklung mit UML und Enterprise Architect
        9. <http://www.oose.de>
        10. <http://www.oose.de/wp-content/uploads/2011/11/oose-UML-Toolliste.pdf>