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

Gliederung

1. Einleitung
   1. Grundlagen des Projektmanagements
      1. Definition
      2. Modelle
      3. Phasen
      4. Prozesse
2. UML
   * 1. Definition
     2. Historie
     3. Struktur
     4. Abstraktionsgrad
     5. Strukturdiagramme
        1. Verteilungsdiagramm
        2. Komponentendiagramm
        3. Paketdiagramm
        4. Klassendiagramm
     6. Dynamische Diagramme
        1. Anwendungsfalldiagramm
        2. Aktivitätsdiagramm
        3. Zustandsdiagramm
     7. Anwendung
     8. Werkzeuge
3. Zusammenfassung
   1. Fazit
   2. Kritische Betrachtung
   3. Ausblick
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, einen Überblick über die Anwendungsmöglichkeiten geben und ein Verständnis schaffen, welches die Stärken und Schwächen beim Einsatz von UML im Prozess der Softwaremodellierung im Kontext des Projektmanagement sind.

Zunächst werden in diesem Kapitel die Grundlagen des Projektmanagements in Bezug auf Softwareprojekte vorgestellt. Insbesondere das Verinnerlichen der einzelnen Phasen der Softwareentwicklung und der verschiedenen Prozessmodelle im Bereich der Analyse und des Designs sind wichtig. Die UML ist für die Modell-Darstellung eines Software-Systems oder Teilen davon und dem darauf folgenden Entwurf der Architektur optimiert.

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 neben einer Zusammenfassung, eine kritische Betrachtung der UML und ein Ausblick angefügt.

* 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 zu können, 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 geplante 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 wesentlich ist, ist abhängig davon, welche Ziele mit dem Modell verfolgt werden und wer mit dem Modell arbeiten bzw. kommunizieren möchte.

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 lesbar. Nur wenn die verschiedenen Informationen auf einzelne Landkarten verteilt werden, kann man die Karten lesen. So werden verschiedene Sichten auf ein und den selben Gegenstand gebildet. Diese Sichten sind miteinander in vielfältiger Weise verbunden. Wenn eine Sicht geändert wird, müssen in der Regel auch alle anderen Sichten angepasst werden. Wenn sich ein sich ein Grenzverlauf zwischen zwei Ländern ändert, müssen alle Sichten, also alle Landkarten, angepasst werden.

Das Modell eines Software-Systems soll im Rahmen eines Projektes in der Regel folgende Aufgaben erfüllen: Es muss die Kommunikation zwischen denjenigen ermöglichen, die an dem Projekt arbeiten. Dabei ist es unverzichtbar, dass alle Beteiligten ein gemeinsames Verständnis des Systems haben. Die verwendete Notation muss von allen verstanden werden. Auftraggeber und Auftragnehmer müssen sich auf gemeinsame Anforderungen einigen und die Entwickler müssen diese Anforderungen verstehen können. Es muss sichergestellt werden, dass gefällte Entscheidungen auch nach Monaten noch nachvollzogen werden können. Bei all diesen Punkten geht es um die Kommunikation zwischen dem Projektteam und allen Stakeholdern. Dies ist ohne Aufzeichnungen der festgelegten Fakten nicht vorstellbar. Das Sammeln der Fakten und die anschließender Darstellung ist durch ein grafisches Modell einfacher als durch rein textliche Beschreibungen zu erreichen.

Ein Modell ermöglicht es dann, die Fakten in Bezug auf Vollständigkeit, Widerspruchsfreiheit und Korrektheit zu überprüfen. Durch die klare Darstellung insbesondere von Zusammenhängen, ist es möglich, gezielt Fragen zu stellen und zu beantworten.

* + 1. Projektphasen

Von der Initiierung eines Projekts bis zu seinem erfolgreichen Abschluss durchläuft es verschiedene Projektphasen. Die Phasen bilden den zeitlichen Ablauf für das Projektmanagement. Mit der Ausarbeitung der Projektziele und -inhalte wird begonnen. Daran schließt sich die Planung von Arbeitspaketen mit Terminen, Kosten etc. an. Nach dieser Planungsphase sind die Aufgaben während der Umsetzung zu kontrollieren und zu steuern und je nach Notwendigkeit Plananpassungen vorzunehmen. Eine Auswertung zum Abschluss eines Projekts dokumentieren den Projektablauf. Die folgenden Phasen können somit festgehalten werden:

Start -> Planung -> Kontrolle -> Abschluss

Die UML ist vor allem für die Aufgaben der Planungsphase - für die Analyse und das Design eines Software-Systems – gedacht. Es gibt drei wichtige Ziele dieser Phase: Die Festlegung der Grenzen und der Schnittstellen zur Umwelt des Systems; die Definition der Anforderungen, die das System erfüllen soll; und die Modellierung des Begriffsmodells {6}. Für das Erreichen dieser Ziele gibt es in Bezug des Projektmanagements folgende Prozessmodelle.

* + 1. Prozessmodelle

Folgende Prozessmodelle verwenden UML:

* + - * + Rational Unified Prozess (RUP)
        + Extreme Programming (XP)
        + Scrum
        + Crystal
        + Analyse und Design mit der UML
        + V-Modell

Hinweis: Hier folgen pro Prozessmodell ca. drei Sätze.

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.{3}

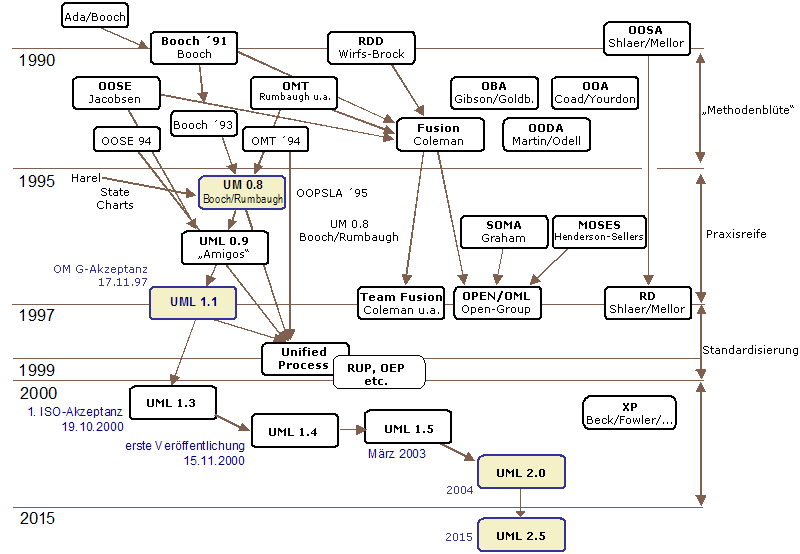
* 1. Historie

Um sich die Ursprünge der UML anzusehen, muss man rund 25 Jahre zurückblicken. Die Thematik der objektorientierten Analyse und des Designs wurde in den 90er Jahren immer wichtiger.

Wie die Abbildung XX zeigt gab es zahlreiche Autoren, die Ansätze lieferten. Doch nur die Methoden von James Rumbaugh und Grady Booch setzten sich durch. {6} Zunächst wollte jeder seine eigene Idee durchsetzen, die jeweils in verschiedenen Breichen ihre Stärke hatte. Rumbaughs Methode schien besser für die Analyse zu sein und Boochs Methode war beim Design vorteilhafter. Erst mit der Hilfe von Ivar Jacobsen versuchten Rumbaugh und Booch dann ihre Differenzen zurückzustellen und zu dritt eine einheitliche Sprache zu schaffen. Dies geschah auch auf Druck der Industrie hin, denn mit entsprechenden Werkzeugen und Vorgehensmodellen auf Grundlage von UML ließ sich Geld verdienen. {3}

Das Ergebnis war die Unified Method (UM). Booch, Rumbaugh und Jacobsen werden seitdem als die drei Amigos bezeichnet. Kurze Zeit wurde die erste Version der UML veröffentlicht. Diese setzte sich als Quasi-Standard durch. 1997 wurde sie in der Version 1.1 bei der Object Management Group (OMG) eingericht und akzeptiert. Die OMG ist ein Indutriekonsortium mit über 800 Mitgliedern, zu denen u.a. IBM, Apple, Microsoft, Oracle, HP, SUN, DaimlerChrysler gehören. Es verwaltet und entwickelt die UML und viele andere Standards weiter. {3}

In kleinen Schritten wurde die Entwicklung bis zur Version 2.0 vorangetrieben. Seit dem Jahr 2000 ist die UML auch ein Standard der ISO, der in der ISO/IEC 19501 spezifiziert. Momentan ist die UML in der Version 2.5 vom Juni 2015 im Downloadbereich der OMG {11} verfügbar.

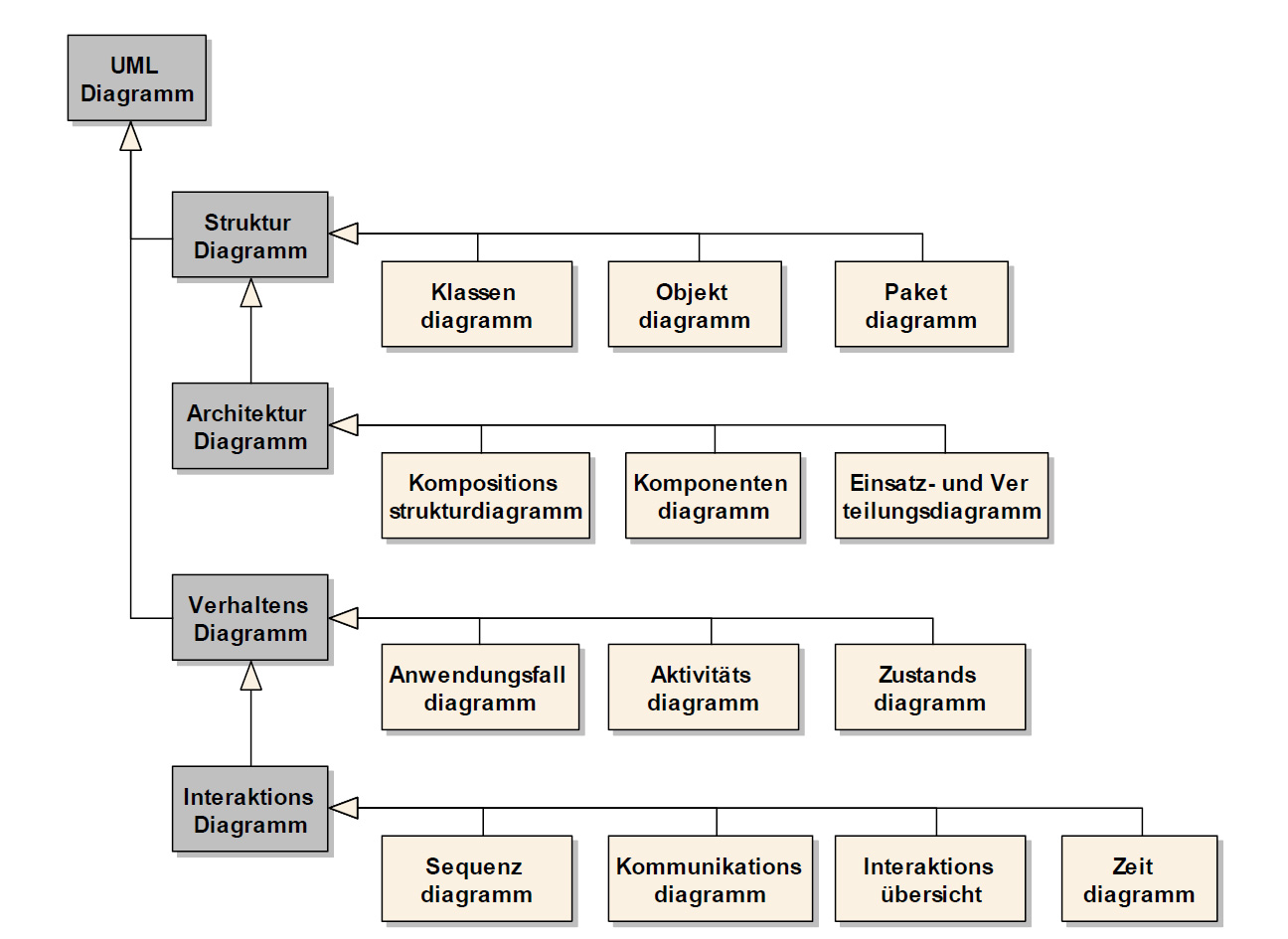


{3} Historische Entwicklung der UML

* 1. Struktur

Die aktuelle UML-Spezifikation der Version 2.5 umfasst 794 Seiten. In ihr werden alle Notationselemente und deren Semantik für die Verwendung in einzelnen Modellen ausführlich beschrieben. Hier findet der Modellbegriff aus Kapitel 1.1.2 seine Umsetzung. Für jede Sicht eines Software-Systems wird ein anderes Modell verwendet. Es gibt in der UML insgesamt 15 Diagrammtypen, die mit ihren eigenen Notationselementen viele verschiedene Sichten auf das System ermöglichen. Die wichtigsten und am häufigsten verwendeten Diagramme werden in Kapitel 2.5 und 2.8 vorgestellt.

Abbildung XX zeigt den systematischen Zusammenhang zwischen den Diagrammtypen, die sich in verschiedenen Gruppen ordenen lasssen. Die Gruppen der Strukturdiagramme und der Architekturdiagramme werden zusammen als statische Diagramme bezeichnet. Verhaltensdiagramme und Interaktionsdiagramme werden als dynamische Diagramme bezeichnet.



{8} Diagramm-Typen

* 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 vom jeweiligen Diagramm zum Software-System 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

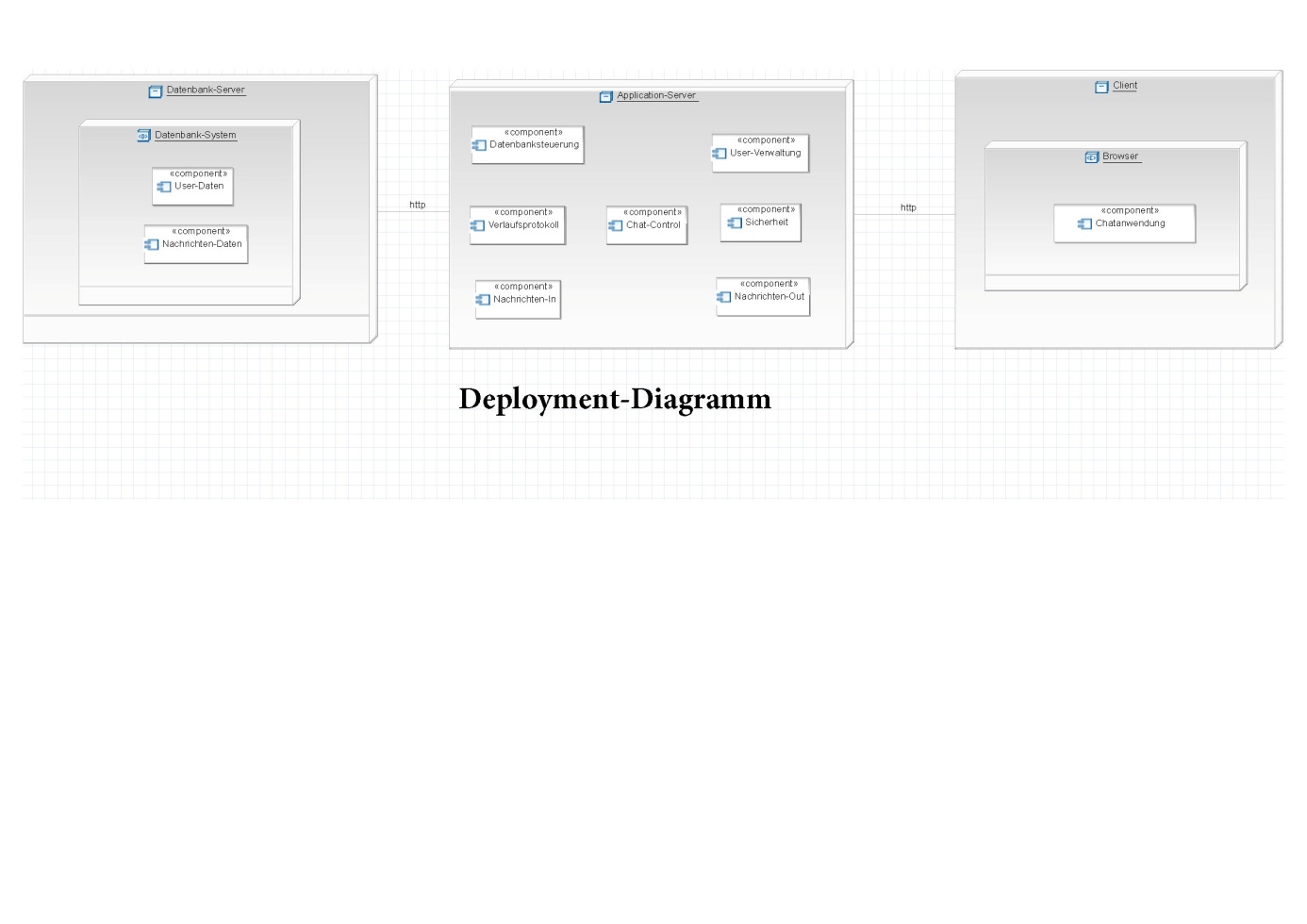
* 1. Struktur-Diagramme
     1. Verteilungs-Diagramm

Das Verteilungs-Diagramm (engl. Deployment Diagram) wird relativ früh spezifiziert. Siehe zweite Abstraktionsebene in Kapitel 2.4. 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 umso wichtiger, je größer die Software-Systeme werden.

Zum Beispiel 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 Software-System-Modells in Form des Verteilungs-Diagrammes ist es, die physische Verteilung aufschlussreich darzustellen.

Zur Veranschaulichung ist in Abbildung XX ein Verteilungs-Diagramm dargestellt, das ein Chatsystem modelliert, das auf einem Application-Server (mitte), einem Datenbank-Server (links) und den Clients (rechts) verteilt ist.



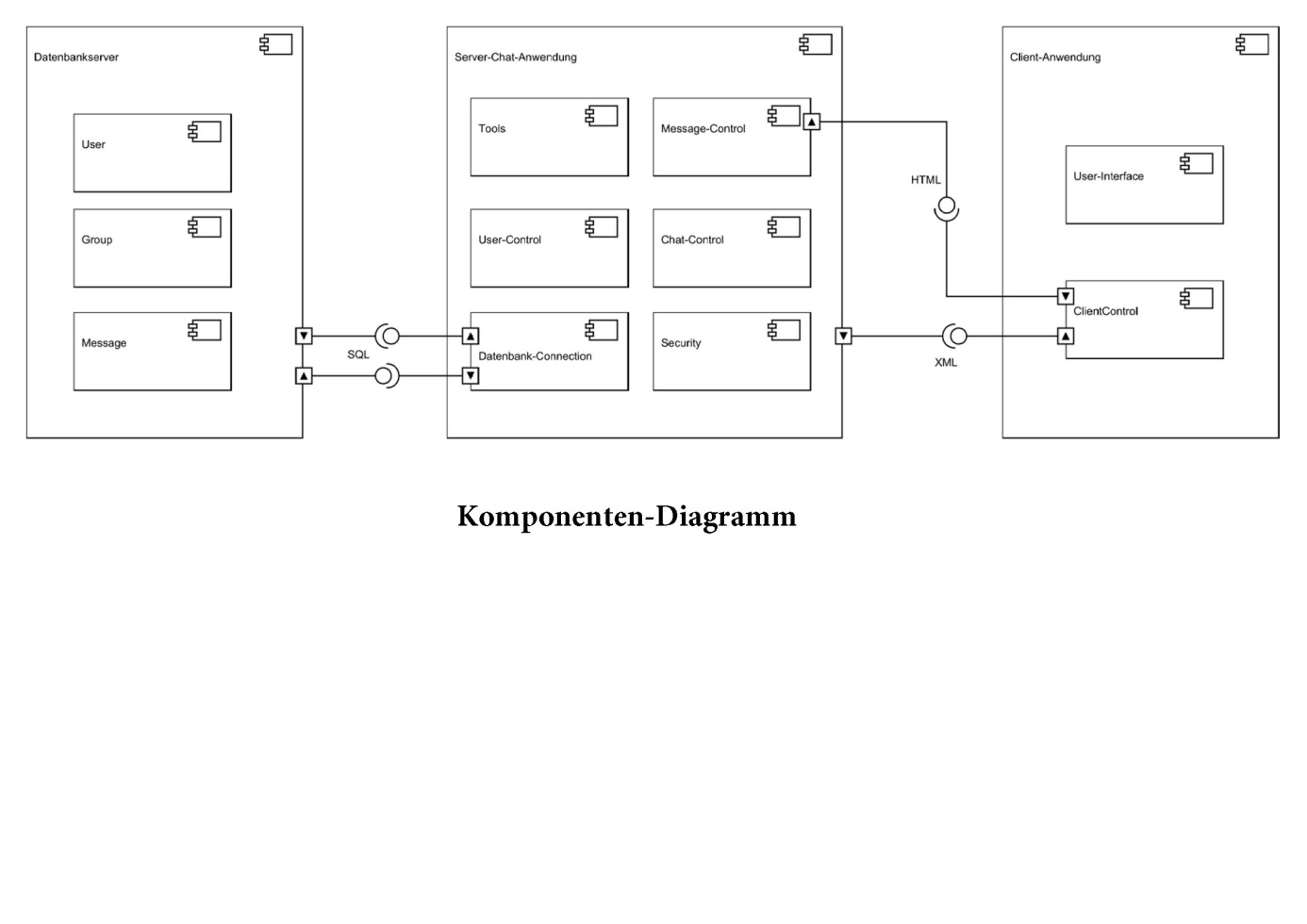
* + 1. Komponenten-Diagramm

Das Komponenten-Diagramm (engl. Component Diagram) beschreibt eine bestimmte Menge von Komponenten. Es soll zeigen, wie das konkrete System strukturiert ist. In der Regel sind es mehrere Klassen, es kann aber auch nur eine Klasse sein, die aber keine Details zeigt.

Auch im Deployment-Diagramm war eine Menge von Komponenten zu sehen, die einzeln oder zusammen eine oder mehrere bestimmte Aufgaben erfüllen.

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

Bernd Oesterreich 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.“{3}



Das Komponenten-Diagramm in Abbildung XX zeigt biespielhaft die Schnittstellen und Kapselung der Klassen zu den Komponenten im Chat-System.

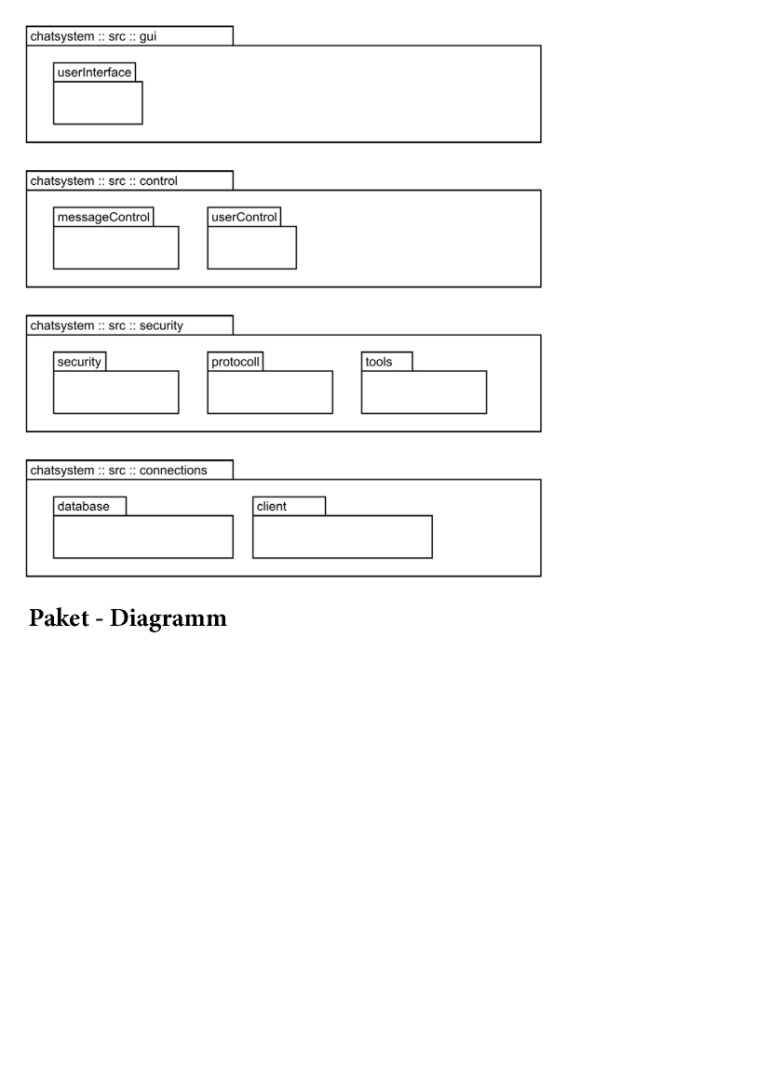
* + 1. Paket-Diagramm

Das Paket-Diagramm strukturiert den sogenannten Namensraum von Paketen, Komponenten oder Klassen, der auch in Abbildung XX am Chat-System bezeichnet ist. Dieser spiegelt die hierarchische Struktur wieder, die die eindeutige Identifizierung des Modellelementes ermöglicht. In der Praxis sind Paket-Diagramme wichtig, weil die Entwickler bereits frühzeitig Testcode entwickeln wollen, der gegliedert werden sollte. Dazu dient das Paketdiagramm.

Pakete können wie Komponenten-Diagramme, die wirkliche Architektur des Systems wiedergeben. Design Patterns und Schichten in der Anwendung werden hier nicht beachtet. Wichtig ist, dass diese Transparenz frühzeitig gegeben ist, von vornerein ausreichend darüber diskutiert wird oder erfahrene Designer zum Einsatz kommen.

Am wichtigsten sind Paket-Diagramme, weil man mit ihnen die Abhängigkeiten modellieren kann.

Beispielsweise kann man in einem Paket-Diagramm, welches benutzte Beziehungen enthält, auch sehr gut erkennen, ob es Zyklen in der Verwendung gibt. Wenn Paket A das Paket B verwendet und umgekehrt, ist meistens ein Redesign nötig.



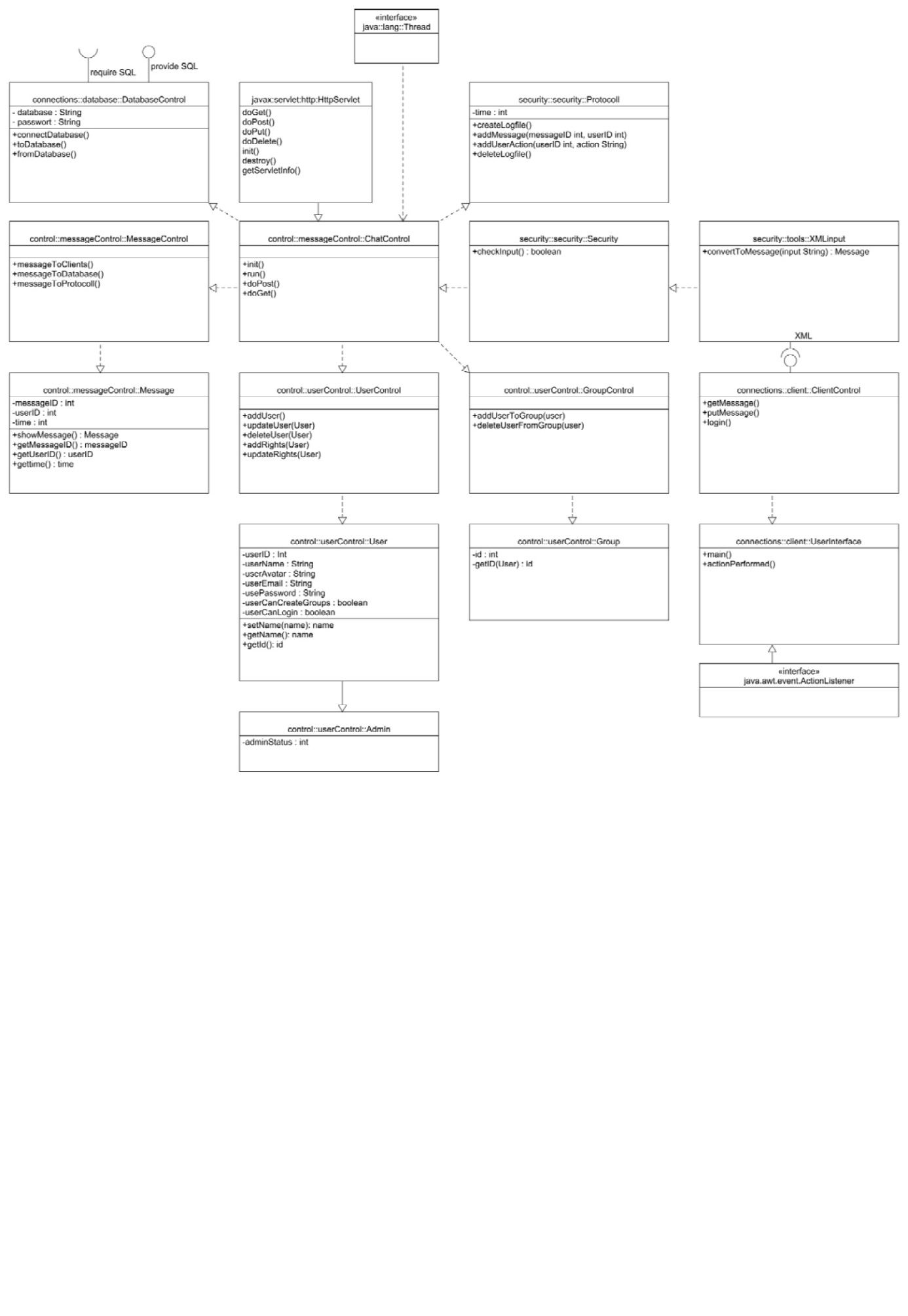
* + 1. Klassen-Diagramm

Ein Klassen-Diagramm visualisiert Klassen auf verschiedene Art und Weise. Dazu werden zunächst die Klassen selbst dargestellt, ggf. auch deren Beziehungen zueinander und evtl. auch deren innnere Struktur. Zwei Ziele werden dabei verfolgt:

Die Darstellung der Klassen und Beziehungen sowie die Darstellung der inneren Struktur. Bei der Darstellung der Klassen und ihrer Beziehungen zueinander 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.

Bei der Darstellung der inneren Struktur wird definiert, welche Attribute oder Methoden die Klassen verwenden.

Beide Formen werden oft miteinander kombiniert und bilden dann ein ausführliches Klassen-Diagramm wie das Beispiel des Chat-Systems in Abbildung XX zeigt.



Klassen-Diagramm des Chat-Systems

* 1. Dynamische Diagramme
     1. Anwendungsfalldiagramm

Wird ergänzt

* + 1. Aktivitätsdiagramm

Wird ergänzt

* + 1. Zustandsdiagramm

Wird ergänzt

* 1. Anwendung

Der eigentliche Sinn der UML liegt darin, im Rahmen der Software-Entwicklung 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 nicht immer das beste Mittel sein muss und auch nicht vollständig sein kann. Daneben gibt es jedoch noch weitere Anwendungsmöglichkeiten, die auch im Kontext des Projektmanagements von Bedeutung sind.

Eine weitere Möglichkeit der UML ist das Reverse-Engineering. Hierbei sei schon ein System gegeben, das weiterentwickelt werden soll und man möchte sich einen Überblick über das System verschaffen. Gute Werkzeuge (siehe Kapitel 2.8.) 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 Komponenten-Diagramme) zeigen zu lassen und so das Programm erst einmal auf der Modellebene 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 die UML oft mehr informell in Teamdiskussionen quasi als Sketch verwendet. Als Sketch (Skizze) wird UML verwendet und die dargestellten UML-Diagramme zeigen meist nur die interessanten Ausschnitte des Systems, das gerade besprochen werden soll.

Im Gegensatz dazu ist der Architekt des Systems im oben genannten Fall damit beschäftigt, das Design der Software mehr oder weniger vollständig zu hinterlegen. Dies wird dann als Blueprint (Blaupause) bezeichnet. Die Verwendung als Sketch und Blueprint sind nicht synonym zu verwenden. Ein Sketch ist eine kleine Diskussion über Ausschnitte. Ein Blueprint ist eine größere Architekturvorgabe.

Die UML kann außerdem selbst als Programmiersprache verwendet werden. Entscheidend ist, dass sie selbstständig 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, dann sollte auch möglichst viel Code auf Knopfdruck automatisch generiert werden. In diesem Fall wäre dann die UML selbst so etwas wie eine höherwertige Meta-Programmiersprache.

Dieses Vorgehen führt zum Forschungsgebiet der MDA, der Model Driven Architecture, wofür es bereits sehr 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. 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

UML hat sich als Best Practise in der Wirtschaft durchgesetzt und wurde ein anerkannter Standard, der sich in den letzten 20 Jahren sehr verbreitet hat. Die universielle Notationselemente bieten sich an, um komplexe reale Systeme in Modellansichten widerzuspiegeln. Es fördert die Kommunikation in fast allen Projektphasen und hilft auch bei Systemdokumentation und –Wartung. Bereits in der Analyse- und Desingphase können mit der Hilfe von der UML Fehler früh behoben werden. Die Verwendung hat sich in der Industrie durchgesetzt. Sie wird an den Hochschulen gelehrt und ist in der Literatur sehr beliebt.

Es gibt einige Vorteile, die für die UML sprechen. Ein modelliertes Softwaresystem-Modell kann in der Implementierungsphase unabhängig von Hardware und Software auf verschiedenen Plattformen umgesetzt werden. UML bietet dazu die nötige Flexibilität, um die Darstellung eines komplexen Systems in vielen verschiedenen Modellsichten zu ermöglichen. So wird immer nur das betrachtet, um das es in der Projektumsetzung gerade geht. Es können Details verborgen werden, um diese dann zu einem späteren Zeitpunkt verfeinern zu können. Außerdem bietet die UML die Möglichkeit, sich mit Personen, die keine IT-Kenntnisse haben, aber UML verstehen können, sich über ein System auszutauschen.

Es gibt eine Reihe an Prozessmodellen, die im Projektmanagement sehr verbreitet sind und die UML als Modellierungsprache verwenden{3}:

* + - * + Rational Unified Prozess (RUP)
        + Extreme Programming (XP)
        + Scrum
        + Crystal
        + Analyse und Design mit der UML
        + V-Modell

Als größten Nachteil kann man anführen, dass für die Verwendung der UML zur Entwicklung von Softwaresystemen eine sehr genaue Kenntnis der UML notwendig ist und ein nicht unerheblicher Aufwand in das Designen der Modelle investiert werden muss.

* 1. Kritische Betrachtung

Die Anwendung und die Bedeutung der UML ist nicht unumstritten.{7} Es gibt Unternehmen und Bewegungen, die die Nützlichkeit von der UML viel geringer bewerten. Gerade auch bei agilen Prozessmodellen wird deshalb oft auf andere Techniken gesetzt. Dies ist bereits beim XP Modell zu sehen, welches keine umfangreichen UML-Diagramme vorsieht. In einigen bedeutenden Unternehmen wie zum Beispiel Google ist UML sogar verboten. Dort wird die Kraft eines Prototypen oder eine prototypische Architektur im Unternehmensverständnis als viel effizienter angesehen als ein UML-Diagramm.

Für jeden Projektmanager ist es deshalb entscheidend, das Spektrum der Werkzeuge oder Kommunikationsmittel zu kennen und selbst beurteilen zu können, inwieweit die UML im Rahmen des gewählten Prozessmodell ein geeignetes Mittel ist oder ob andere Verfahren besser geeignet sind.

* 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.

Agile Modellierung mit UML: das beste aus zwei Welten.

Hier folgt noch eine kurze Ausformulierung

Anmerkungen und To-do:

* + - * + 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>
        11. <http://www.omg.org/spec/UML/2.5/PDF/>