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

Niels Bubel  
mail@niels-bubel.de

Beuth Hochschule für Technik Berlin

Kurs: Einführung in das Projektmanagement  
Dozent: Prof. Dr. Michael Syrjakow

Gliederung

1. Einleitung
   1. Grundlagen des Projektmanagements
      1. Definition
      2. Modell
      3. Phasen
2. Unified Modeling Language
   * 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
     7. Anwendung
     8. Werkzeuge
3. Zusammenfassung
4. Einleitung

Die Unified Modeling Language (UML) ist eine der wichtigsten Methoden, um das Projektmanagement von Software-Systemen zu unterstützen. Diese Arbeit soll in die UML einführen, einen Überblick über die Anwendungsmöglichkeiten geben und die Stärken und Schwächen von UML benennen.

Zunächst werden in der Einleitung die Grundlagen des Projektmanagements für Softwareprojekte vorgestellt. Insbesondere sind hier die einzelnen Phasen der Softwareentwicklung und der verschiedenen Prozessmodelle im Bereich der Analyse und des Designs von Bedeutung. Die UML ist für die Modell-Darstellung eines Software-Systems oder Teilen davon und dem darauf folgenden Entwurf der Softwarearchitektur optimiert.

In Kapitel zwei werden zunächst die Definition der UML und deren Historie beschrieben. Darauf folgt ein Überblick über die Struktur und die Zusammenhänge der wichtigsten Diagramm-Typen der UML. Anhand von sechs verschiedenen Typen werden dabei auch die Zielsetzungen der UML vorgestellt. Eigene Modellierungen mit Diagramm-Typen zeigen beispielhaft unterschiedliche Sichten auf ein selbst entwickeltes Chat-System.

Daran schließen sich Beispiele für die vielseitigen Anwendungsmöglichkeiten der UML an, die über die reine Dokumentation von Projekten hinausgehen und zu automatisierten Code-Generatoren im Rahmen der „Model Driven Architectur“ führen.

Das im abschließende dritte Kapitel stellt eine Zusammenfassung der Arbeit dar.

* 1. Grundlagen des Projektmanagements
     1. Definition

In der Lehre wird unter dem Begriff Projektmanagement Folgendes 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 gilt für alle Arten von Projekten. In dieser Arbeit werden jedoch nur Projekte zur Erstellung von Software betrachtet. Um die Aufgaben, die für die Realisierung solcher Projekte notwendig sind, müssen die Anforderungen an das Software-System zunächst festgestellt, dokumentiert und kommuniziert werden. Dazu werden Modelle benötigt. Diese Modelle bilden das angestrebte Software-System, den zeitlichen Ablauf des Projektmanagements insgesamt und die einzelnen Teile des Entwicklungsprozesses und deren Methoden ab.

* + 1. Modell

Das Modell eines Software-Systems soll im Rahmen eines Projektes hauptsächlich folgende Aufgabe erfüllen: Es muss die Kommunikation zwischen allen am Projekt Beteiligten ermöglichen. Dabei ist es unverzichtbar, dass diese Personen 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 Aspekten geht es um die Kommunikation zwischen dem Projektteam und den Stakeholdern. Dies ist ohne detaillierte Aufzeichnungen nicht vorstellbar. Diese Aufzeichnungen und deren anschließende Darstellung gelingt durch ein grafisches Modell einfacher und präziser als durch rein textliche Beschreibungen. {7}

Ein solches Modell ermöglicht es dann, das Software-System 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 Phasen, die in Abbildung XX dargesteltt sind. 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, Risiken etc. an. Nach der Planungsphase sind die Aufgaben während der Durchführung zu kontrollieren und zu steuern und je nach Notwendigkeit Anpassungen am Plan vorzunehmen {1}. Dabei ist nicht nur die Kommunikation innerhalb des Projektteams, sondern auch die Einbindung der Stakeholder von Relevanz. Eine Auswertung zum Abschluss eines Projekts dokumentiert den Projektablauf. Als Phasen können somit festgehalten werden:

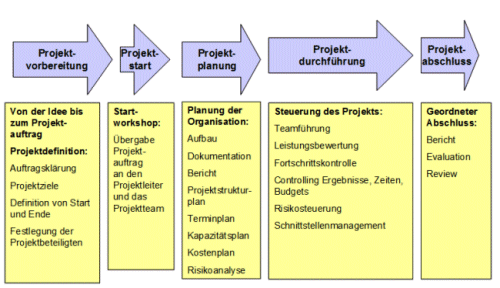


Abbildung XX: Projektphasen

Modelle werden für die Aufgaben der Planungsphase - für die Analyse und das Design eines Software-Systems – un deren anschließende Durchführungsphase verwendet. In diesen Phasen gibt es drei wichtige Ziele, die die Entwicklung des Software-Systems direkt betreffen: die Festlegung der Grenzen und der Schnittstellen zur Umwelt des Softwaresystems, die Definition der zu erfüllenden Anforderungen und die Modellierung der Elemente des Software-Systems {6}.

Für das Erreichen dieser Ziele gibt es eine Reihe von Prozessmodellen. Unter der Vielzahl der Prozessmodelle verwenden die folgenden die UML: der Rational Unified Prozess (RUP), das Extreme Programming (XP), Scrum, Crystal, die Analyse und Design mit der UML, das V-Modell {3}.

1. Unified Modeling Language
   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

Die Ursprünge der UML lagen in den 90er Jahren. Damals wurde die Thematik der objektorientierten Analyse und des Designs immer wichtiger.

Wie die Abbildung XX zeigt gab es zahlreiche Autoren, die Ansätze lieferten. Doch nur die Methoden von James Rumbaugh und Grady Boochsetzten sich durch.{6} Die verschiedenen Ansätze unterschieden sich, Rumbaughs Methode schien besser für die Analyse zu sein und Boochs Methode war beim Design vorteilhafter. Erst mit der Hilfe von Ivar Jacobsen gelang es Rumbaugh und Booch ihre unterschiedlichen Ansätze zu vereinen und eine einheitliche Sprache zu schaffen. Dies geschah auch auf Drängen der Industrie hin, die entsprechende Werkzeuge forderte {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 zunächst 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 ist. Momentan ist die UML in der Version 2.5 vom Juni 2015 im Downloadbereich der OMG {11} verfügbar {3}.

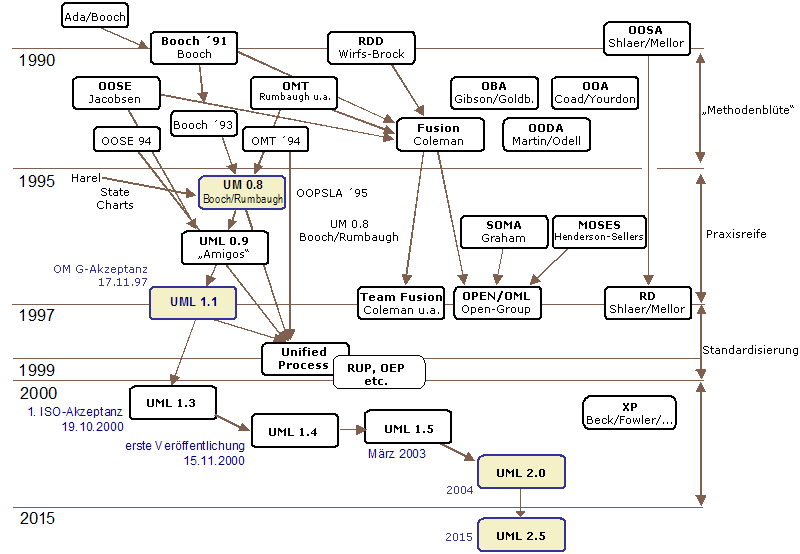


Abbildung XX: Entwicklungsgeschichte der UML, Quelle: {3}

* 1. Struktur

Die aktuelle UML-Spezifikation in der Version 2.5 umfasst 794 Seiten {14}. In ihr werden alle Notationselemente und deren Semantik für die Verwendung der unterschiedlichen Modelle des Projekts ausführlich beschrieben, die unterschiedlichen Sichtweisen auf das Projekt entsprechen. Für unterschiedliche Sichtweisen auf ein Software-System stehen in der UML insgesamt 15 Diagrammtypen mit jeweils eigenen Notationselementen zur Verfügung. Eine Auswahl der wichtigsten und am häufigsten verwendeten Diagramme werden in den Kapiteln 2.5 und 2.6 vorgestellt. Dort werden aufgrund der Komplexität der Notationselemente, die in einem Metamodell {13} visualisiert und für die Praxis in einer anwendungsnahen Übersicht {12} dargestellt werden, der Fokus auf die Ziele der Diagrammtypen gelegt.

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

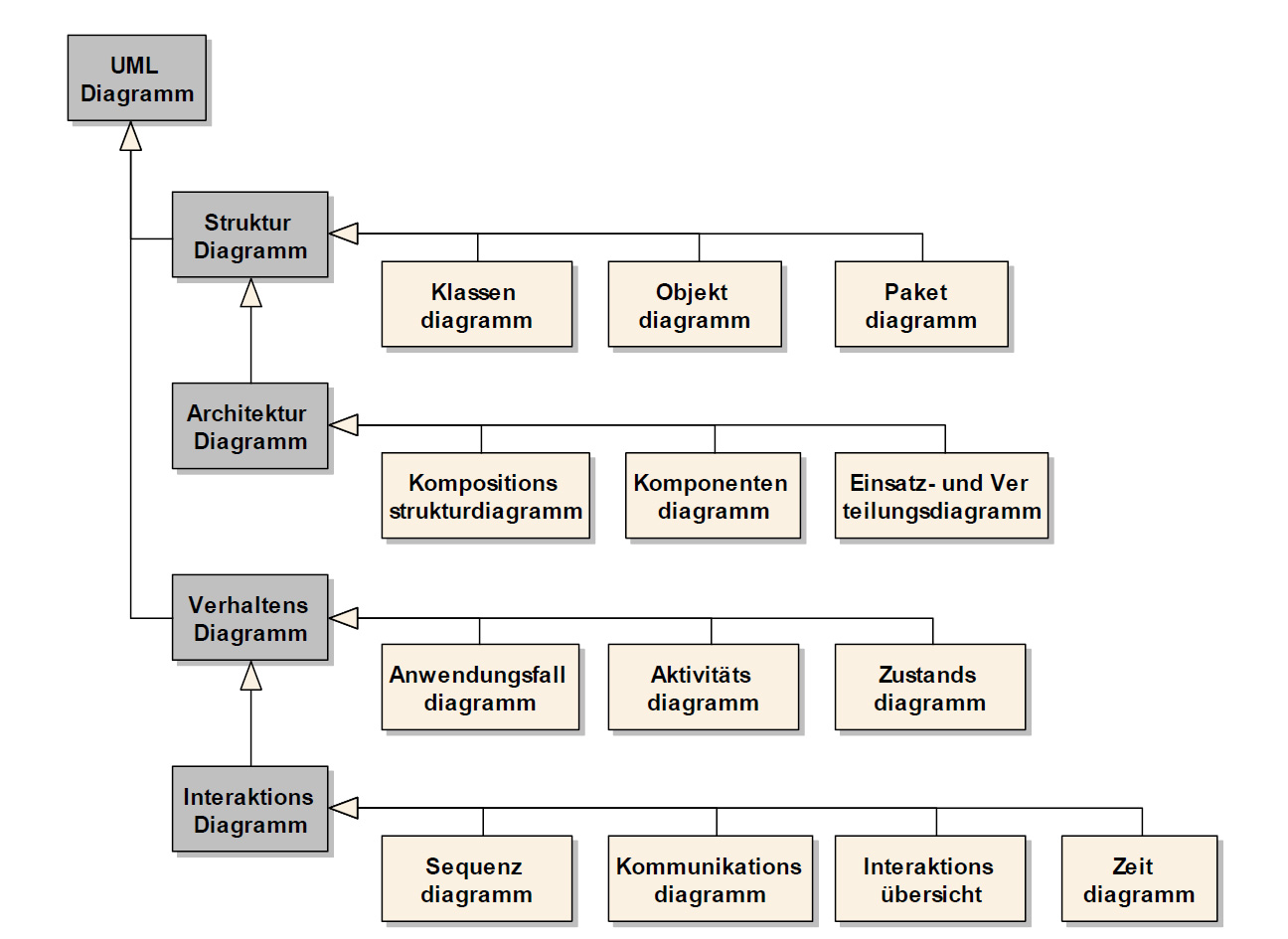


Abbildung XX: Diagramm-Typen, Quelle: {8}

* 1. Abstraktionsgrad

Um die Vielzahl der UML-Diagramm-Typen systematisieren und zugleich eine sinnvolle Reihenfolge bei der Projektarbeit vorzugeben, orientiert man sich zweckmäßigerweise am Abstraktionsgrad der Diagramme. Von den 15 Typen werden im Folgenden sechs exemplarisch ausgewählt und im Kapitel 2.5 vorgestellt.

Die sechs Diagrammtypen können nach fünf Ebenen unterschieden werden. Der Abstraktionsgrad nimmt von Ebene eins bis fünf immer weiter ab. Dass heißt, dass in der ersten Ebene der Grad der Vereinfachung und damit die „Entfernung“ des jeweiligen Diagramms 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 auf der ersten Abstraktionsebene mit Hilfe des „Anwendungsfall- und des Aktivitäts-Diagramm“ ausgetauscht werden.

Jede Ebene zeigt in Abhängigkeit vom jeweiligen Abstraktionsgrad eine andere Sicht des Systems und verfolgt damit unterschiedliche Ziele. Erst wenn die Anforderungen an das System festgelegt wurden, macht es Sinn in der zweiten Ebene das Einsatz- und Verteilungs-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 auf einer höheren Ebene geändert, so muss auf allen niedrigeren Ebenen geprüft werden, ob dies auch dort eine Änderung zur Folge hat. Eine andere Vorgehensweise ist nicht sinnvoll. Hält man sich beim Modellieren der UML-Diagramme an diese aufgezeigte Reihenfolge, dann werden nachträgliche Änderungen vermieden.

* 1. Struktur-Diagramme
     1. Verteilungs-Diagramm

Das Verteilungs-Diagramm 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 betrachteten Software-Systeme werden sollen.

Zum Beispiel gibt es bei Toll Collect, das System mit dem die LKW-Maut auf deutschen Autobahnen erfasst wird, verschiedene Rechenzentren, Satelliten, Geräte auf Autobahnen und in Raststätten. Es läuft also ganz unterschiedliche Software auf sehr verschiedenen Rechnern an unterschiedlichen Stellen.

Das Ziel des Software-System-Modells in Form des Verteilungs-Diagrammes ist es, die physische Verteilung aufschlussreich darzustellen. {3}

Zur Veranschaulichung einzelner Diagrammtypen wird im Folgenden ein selbst entwickeltes Chatsystem verwendet. In Abbildung XX ist ein Verteilungs-Diagramm dargestellt, das dieses Chatsystem modelliert, welches auf einen Application-Server (Mitte), eine Datenbank-Server (links) und den Clients (rechts) verteilt ist.

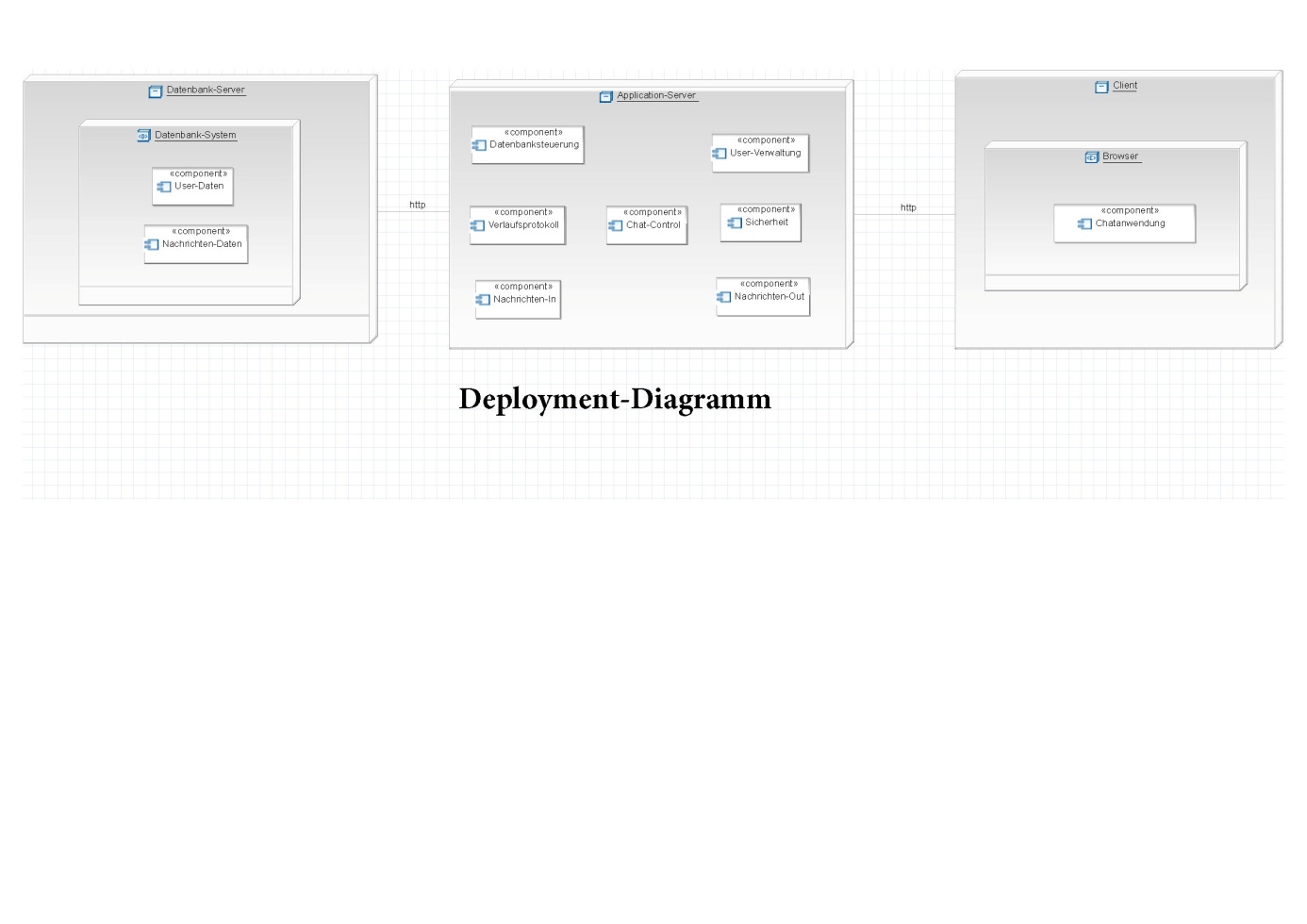


Abbildung XX: Verteilungs-Diagramm eines selbst entwickelten Chat-Systems

* + 1. Komponenten-Diagramm

Das Komponenten-Diagramm beschreibt eine bestimmte Zahl 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 keine Details zeigt.

Auch im Verteilungs-Diagramm sind 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 {3}.

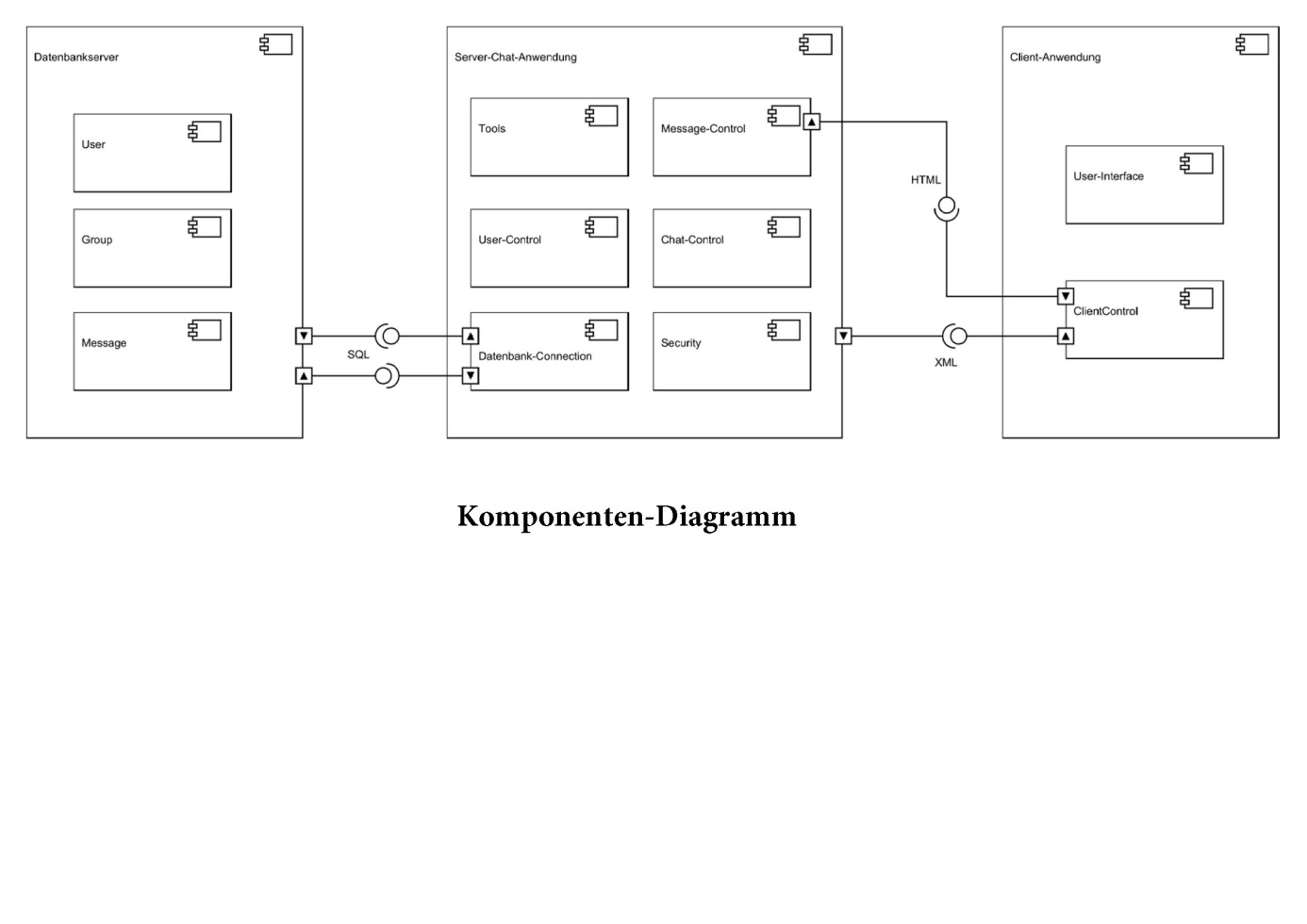


Abbildung XX: Komponenten-Diagramm eines selbst entwickelten Chat-Systems

Das Komponenten-Diagramm in Abbildung XX zeigt biespielhaft die Schnittstellen und Kapselung der Klassen zu den Komponenten in dem bereits mit den anderen Diagrammtypen modellierten Chat-System.

* + 1. Paket-Diagramm

Das Paket-Diagramm strukturiert den sogenannten Namensraum von Paketen, Komponenten oder Klassen, der in Abbildung XX am Beispiel des Chat-Systems wiedergegeben wird. 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 mit ihnen Transparenz frühzeitig gegeben ist, eine Diskussionsbasis verfügbar wird und dabei erfahrene Designer zum Einsatz kommen {6}.

Die größte Bedeutung erlangen Paket-Diagramme, weil man mit ihnen Abhängigkeiten modellieren kann {3}.

Beispielsweise kann man in einem Paket-Diagramm, welches die genutzten 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 {6}.

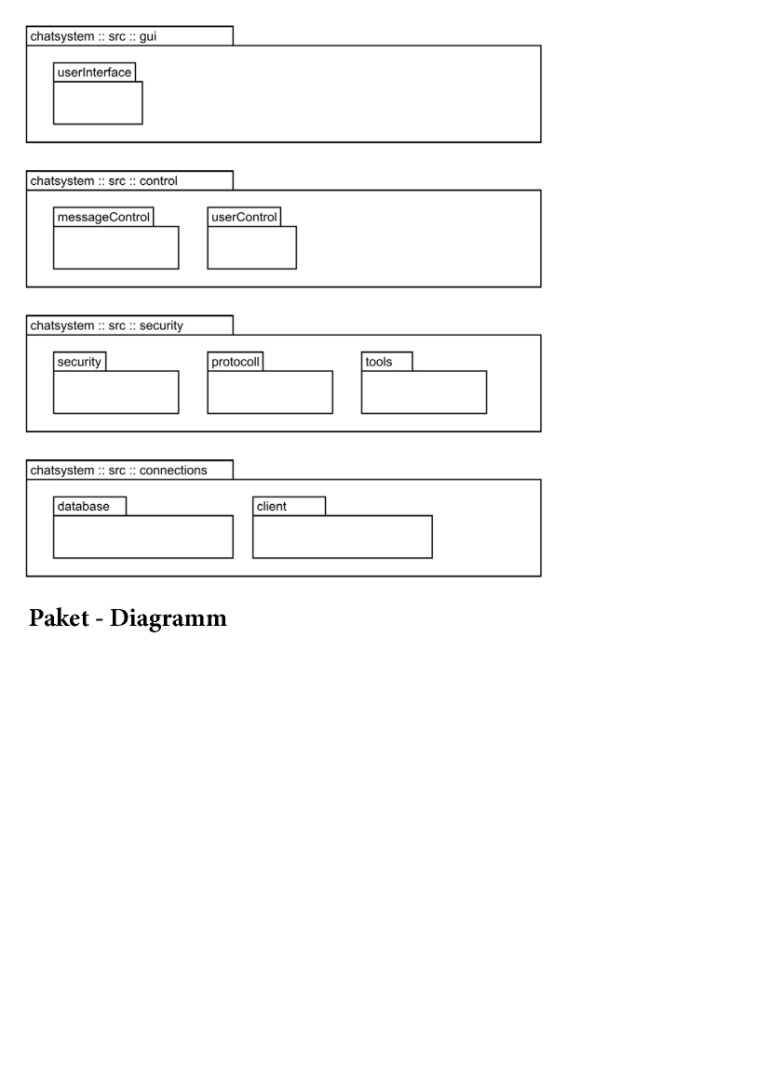


Abbildung XX: Paket-Diagramm eines selbst entwickelten Chat-Systems

* + 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 {7}.

Bei der Darstellung der inneren Struktur wird definiert, welche Attribute oder Methoden die Klassen verwenden {3}.

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

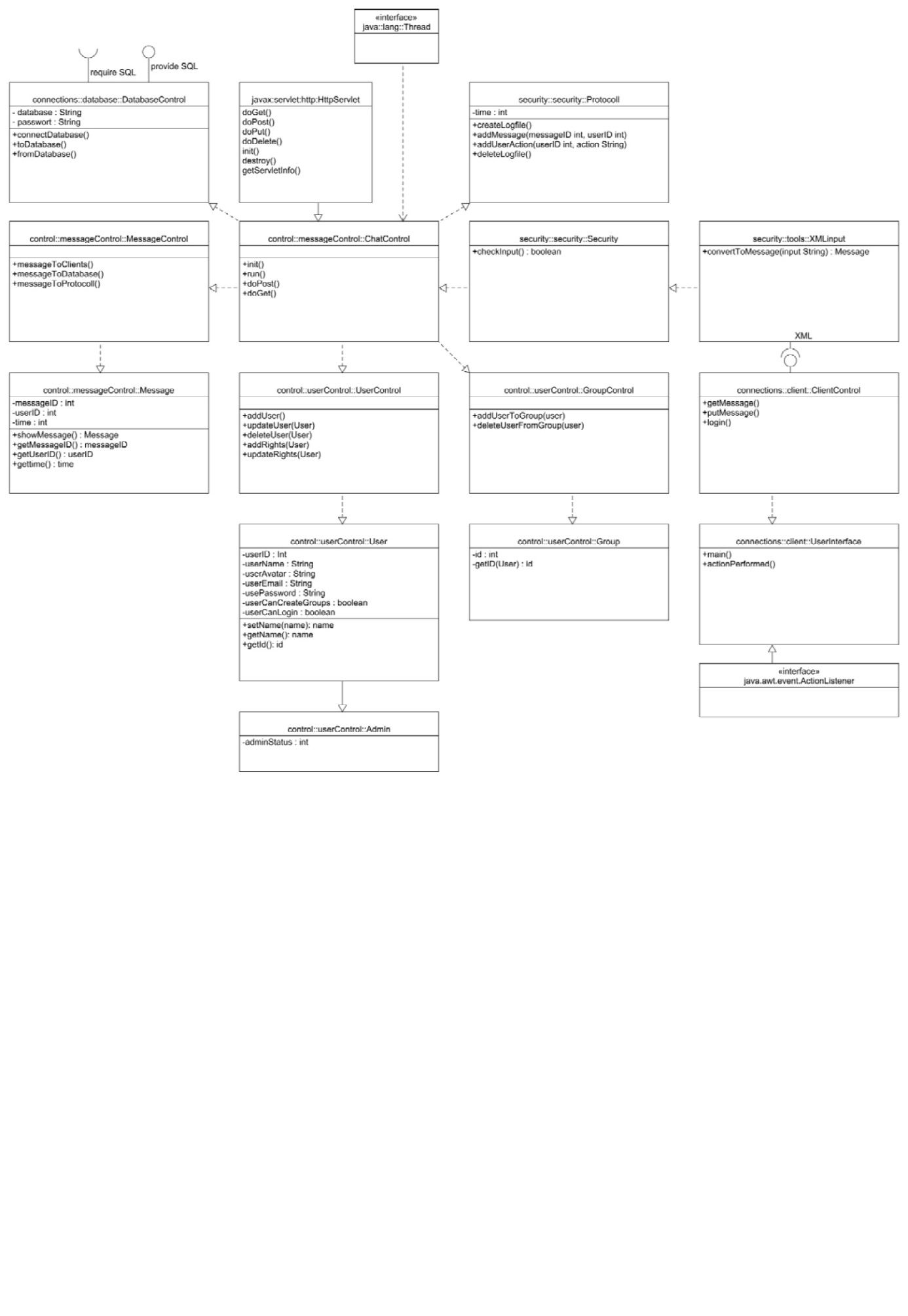


Abbildung XX: Klassen-Diagramm eines selbst entwickelten Chat-Systems

* 1. Dynamische Diagramme
     1. Anwendungsfalldiagramm

Anwendungsfalldiagramme gehören zu den Dynamischen Diagrammen. Sie werden sowohl im Englischen als auch im Deutschen als Use-Case Diagramme bezeichnet. Solche Diagramme sind wichtig und sollten das erste UML-Diagramm sein, welches erarbeitet wird {7}.

Ziel dabei ist es, möglichst einfach zu zeigen, wozu man das zu bauende Softwaresystem benutzen möchte. Alle Fälle der Anwendung sollen erfasst werden. In der Analysephase werden diese Anwendungsfälle dokumentiert. Mit einem UML soll dies mit einfachen Grafiken unterstützt werden {6}.

Das Beispiel des in Abbildung XX gezeigten Anwendungsfall-Diagramms des selbst entwickelten Chat-Systems zeigt die sogenannten Akteure als Strichmännchen. Diese können beispielsweise Personen wie Kunden, Administratoren oder beliebige andere Anwender sein.

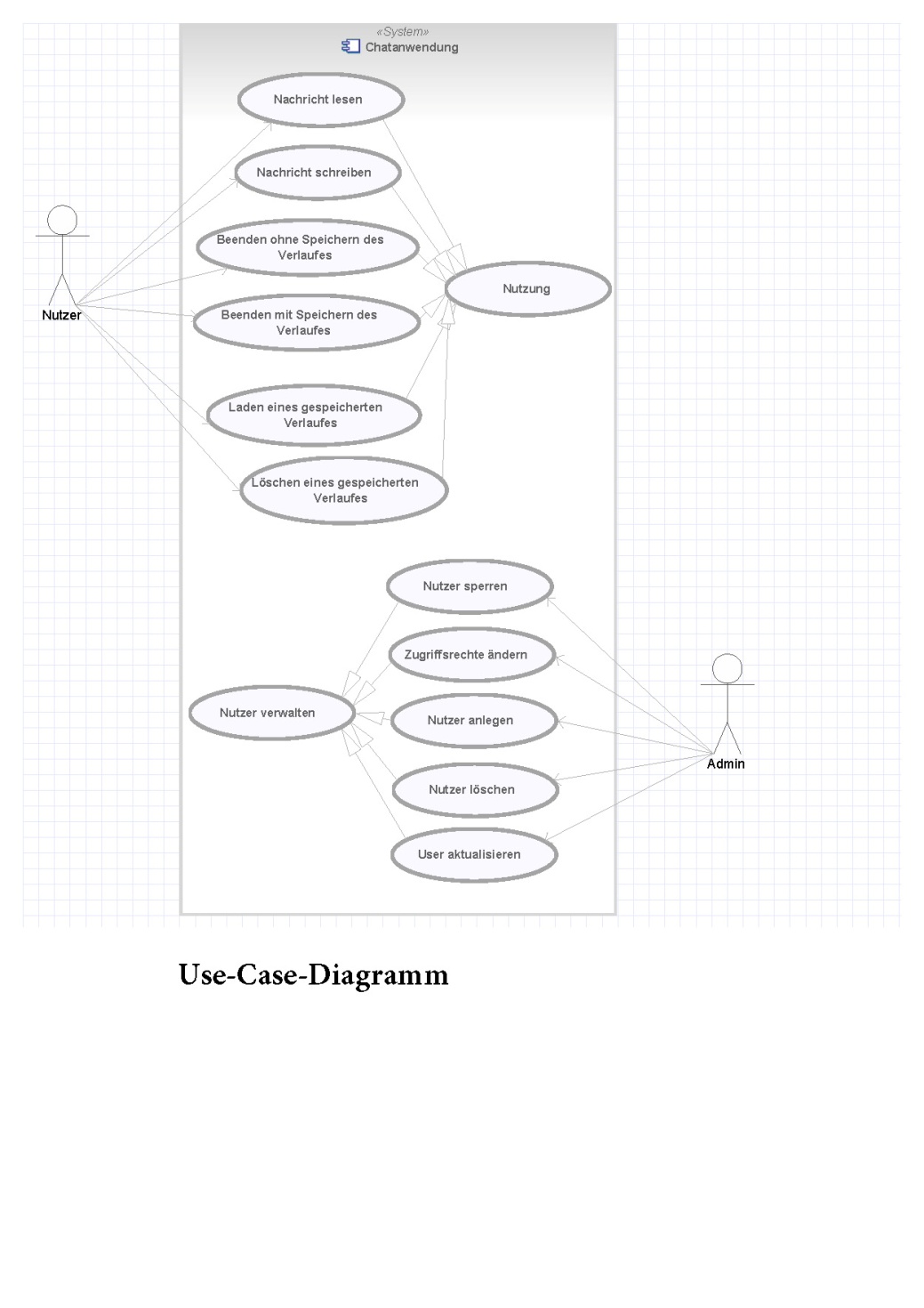


Abbildung XX: Anwendungsfall-Diagramm des selbst entwickelten Chat-Systems

* + 1. Aktivitätsdiagramm

Das Aktivitätsdiagramm ist mit seinen zahlreichen Elementen eines der komplexesten Diagramme. Es wird meist recht früh in der Designphase angewendet und zeigt die Abfolge, in der Anwender durch mögliche Wege der Anwendung geleitet werden. Aktivitätsdiagramme visualisieren und erklären den Ablauf des Programmes, der oft mit der natürlichen Sprache oder dem geschriebenen Wort nur sehr schwer dargestellt werden kann. {3}

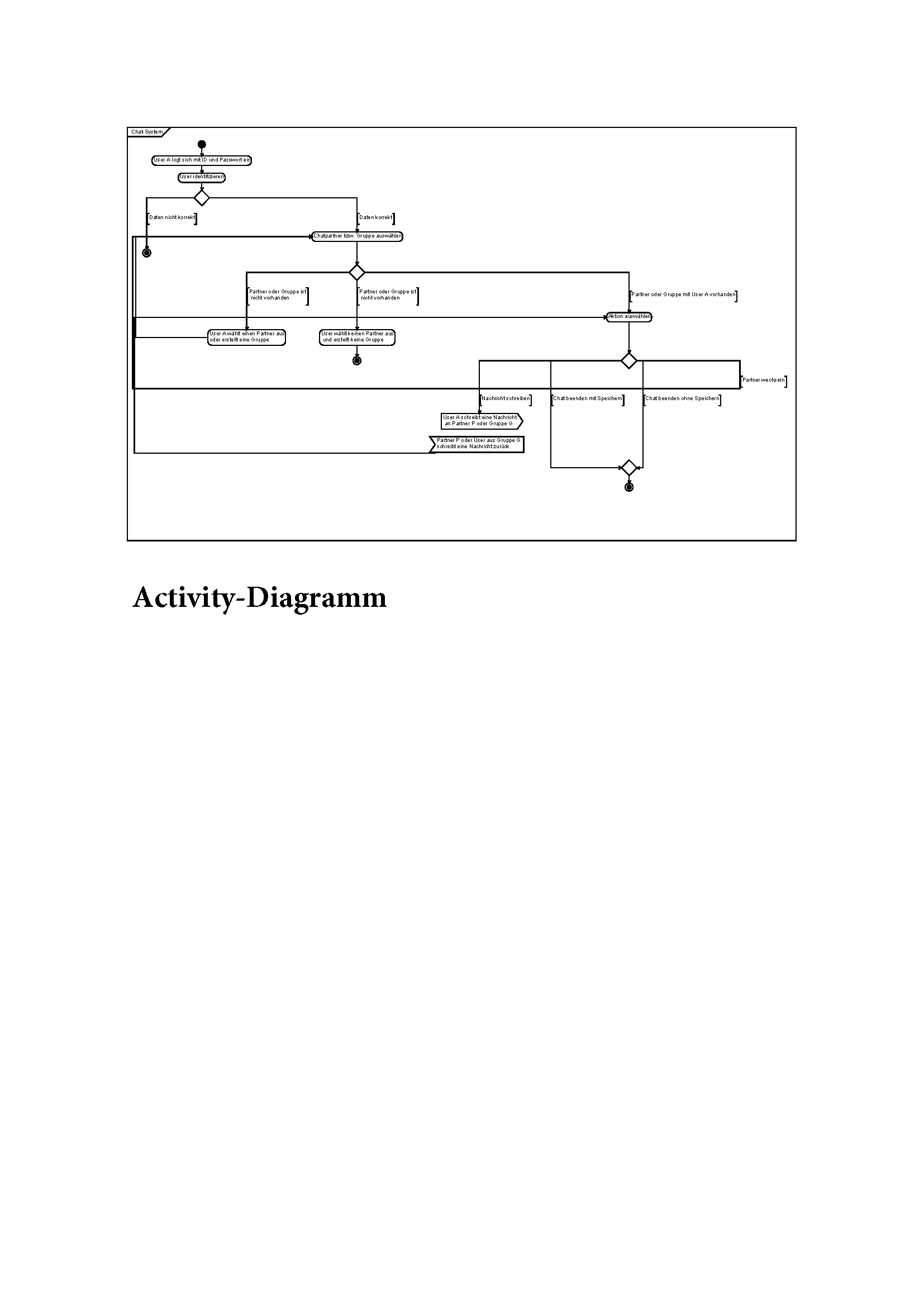


Abbildung XX: Aktivitäts-Diagramm des selbst entwickelten Chat-Systems

* 1. Anwendung

Der entscheidende Vorteil der UML liegt darin, dass alle Beteiligte im Rahmen der Software-Entwicklung kommunizieren. Es stellt sich bei Projektbeginn immer die Frage, wie am besten und schnellsten das Design eines Systems erarbeitet und anschließend dokumentiert wird. Es ist klar, dass UML nicht immer das beste Mittel sein muss und auch nicht vollständig sein kann.

Neben der schriftlichen Dokumentation einer Architektur wird die UML oft auch mehr informell in Teamdiskussionen als Hilfsmittel, als Skizze verwendet. Die dabei verwendeten UML-Diagramme zeigen meist nur die interessanten Ausschnitte des Systems.

Im Gegensatz dazu ist der Architekt des Systems damit beschäftigt, das Design der Software mehr oder weniger vollständig zu hinterlegen. Dies wird dann als Blueprint (Blaupause) bezeichnet. Ein Blueprint ist also eine größere Architekturvorgabe.

Daneben gibt es jedoch noch weitere Anwendungsmöglichkeiten von UML, die auch im Kontext des Projektmanagements von Bedeutung sind.

Eine solche Möglichkeit der UML ist das Reverse-Engineering. Ausgangspunkt ist ein schon bestehendes System, das weiterentwickelt werden soll und für das man sich einen Überblick verschaffen möchte. Spezielle 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, um so das Programm erst auf der Modellebene verstehen zu können. 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 „Weiterentwicklung“ der Tabellenkalkulation Lotus 1-2-3. Das Programm wurde von Microsoft „zerlegt“, anschließend so weit verstanden, dass es als MS Excel neu gebaut werden konnte.

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 {5}.

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 {5}.

* 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

UML hat sich als Best Practise in der Wirtschaft durchgesetzt und hat sich als anerkannter Standard in den letzten 20 Jahren weit verbreitet. Die universielle Notationselemente bieten sich an, um komplexe reale Systeme in den gewünschten Modellsichten 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 vermieden werden.

Die Verwendung hat sich in der Industrie durchgesetzt, sie wird an den Hochschulen gelehrt und ist in der Literatur sehr verbreitet.

Deshalb gibt es eine Reihe von Prozessmodellen, die im Projektmanagement sehr verbreitet sind und die UML als Modellierungsprache verwenden {3}. Dazu gehören der Rational Unified Prozess (RUP), das Extreme Programming (XP), Scrum, Crystal, Analyse und Design mit der UML und das V-Modell.

Ein modelliertes Softwaresystem-Modell kann in der Implementierungsphase unabhängig von Hardware und Software auf verschiedenen Plattformen umgesetzt werden. UML bietet dazu den nötigen Umfang an Notationselementen, um die Darstellung eines komplexen Systems in vielen verschiedenen Modellsichten zu ermöglichen. So wird immer nur das betrachtet, das für die Projektumsetzung gerade relevant ist. 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, aber möglicherweise UML-Kenntnisse haben, sich über ein zu entwickelndes System auszutauschen.

Als Nachteil von UML kann angeführt werden, dass deren Verwendung 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.

Die Arbeit mit der UML könnte in Zukunft deutlich flexibler und auch vereinfacht werden, wenn die Hersteller der UML-Tools sich an der neuen Spezifikation 2.5 aus dem Sommer 2015 orientieren und einen gemeinsamen Standard für den UML-Diagramm-Austausch schaffen würden. Damit wäre die Modellierung mit verschiedenen Tools möglich, so dass deren verschiedene Spezialisierungen kombiniert werden könnten, und keine Abhängigkeit an ein bestimmtes Dateiformat mehr bestünden.

Die Anwendung und die Bedeutung der UML findet jedoch auch Kritik {7}. Es gibt Unternehmen und Bewegungen, die die Nützlichkeit der UML als gering bewerten. Gerade bei agilen Prozessmodellen wird deshalb auf andere Techniken gesetzt. Dies wird bereits beim XP Modell deutlich, welches zumindest 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.

Eine aufkommende Entwicklung zeigt aber auch, dass agile Prozessmodelle mit der UML gut vereinbar sind. Es zeigt sich, dass die Stärken der agilen Modellierung zu Synergien führen, biespielsweise bei der Model Driven Architecture (MDA) {5}. Dabei wird die UML neben der Dokumentation von Entwürfen zur automatisierten Generierung von Code und zur späteren Verbesserung mittels Refactoring-Methoden genutzt. Damit wird die Effizienz in einem Projekt gesteigert und das primäre Ziel, ein laufffähiges Software-System zu entwicklen, unterstützt.

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

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/>
        12. <http://www.oose.de/downloads/uml-2-Notationsuebersicht-oose.de.pdf>
        13. <http://www.oose.de/wp-content/uploads/2013/02/UML-2.4.1-Taxonomy.pdf>
        14. <http://www.omg.org/spec/UML/2.5/PDF/>