Ich begrüße Sie zur letzten Einheit aus Objektorientierter Modellierung. Wir kommen zum leichtesten oder zum schwierigsten Diagramm – je nachdem, wie man es nimmt. Nämlich von den Notationsmöglichkeiten, was ich alles ausdrücken kann, ist es das einfachste Diagramm, es richtig einzusetzen, ist fast am schwierigsten.

Eigentlich bin ich fast überfordert in dem Sinne in einer Vorlesung mit eineinhalb Stunden das Anwendungsfalldiagramm ungefähr zu vermitteln. Ich glaube, dass gerade das Anwendungsfalldiagramm nachher ein Diagramm ist, wo man sehr viel Erfahrung im Software Engineering Prozess braucht, damit man es auch richtig einsetzen kann.

Und dementsprechend ist es eher das methodischen Vorgehen, d.h. wie ich Anwendungsfalldiagramme einsetze, interessant und nicht so sehr die Syntax. In dieser Lehrveranstaltung Objektorientierte Modellierung haben wir ja Großteils die Syntax der verschiedenen Diagramme kennengelernt und weniger den Entwicklungsprozess, den dahinterstehenden Software Engineering Prozess. Hier probiere ich ein bisschen das Vorgehen auch zu erläutern, weil sonst nicht sehr viel übrig bleibt außer Ellipsen und Strichmanderln. Wir wollen ein bisschen versuchen zu sehen, dass noch mehr hinter den Anwendungsfalldiagrammen steckt.

Literatur

- Die Vorlesung basiert auf folgendem Buch:

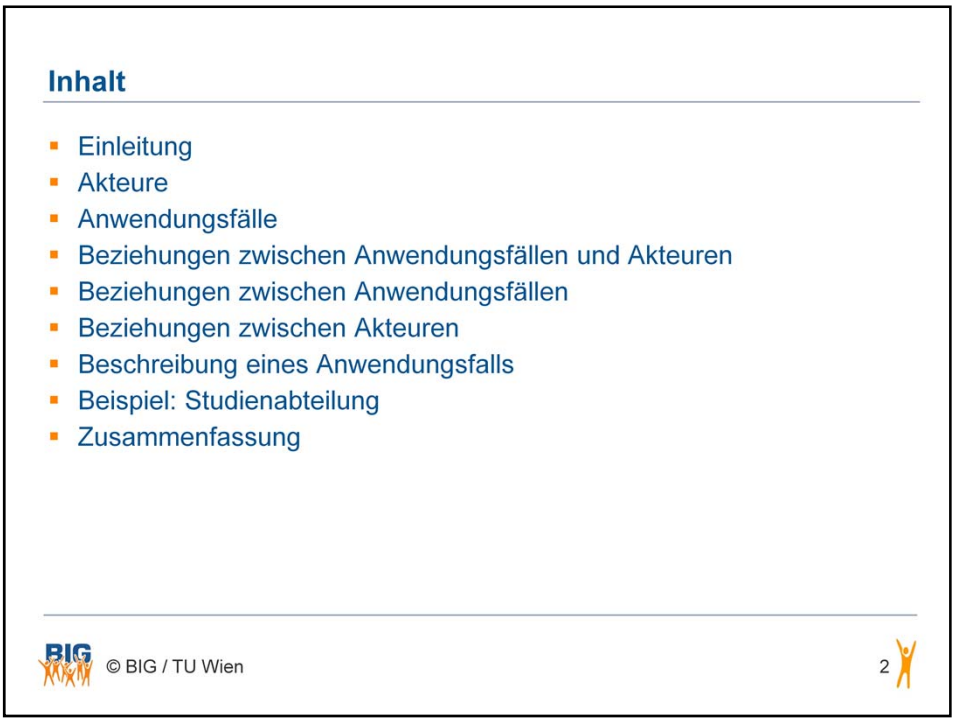
UML @ Classroom:
Eine Einführung in die objekt-orientierte Modellierung
Martina Seidl, Marion Brandsteidl, Christian Huemer und Gerti Kappel

dpunkt.verlag

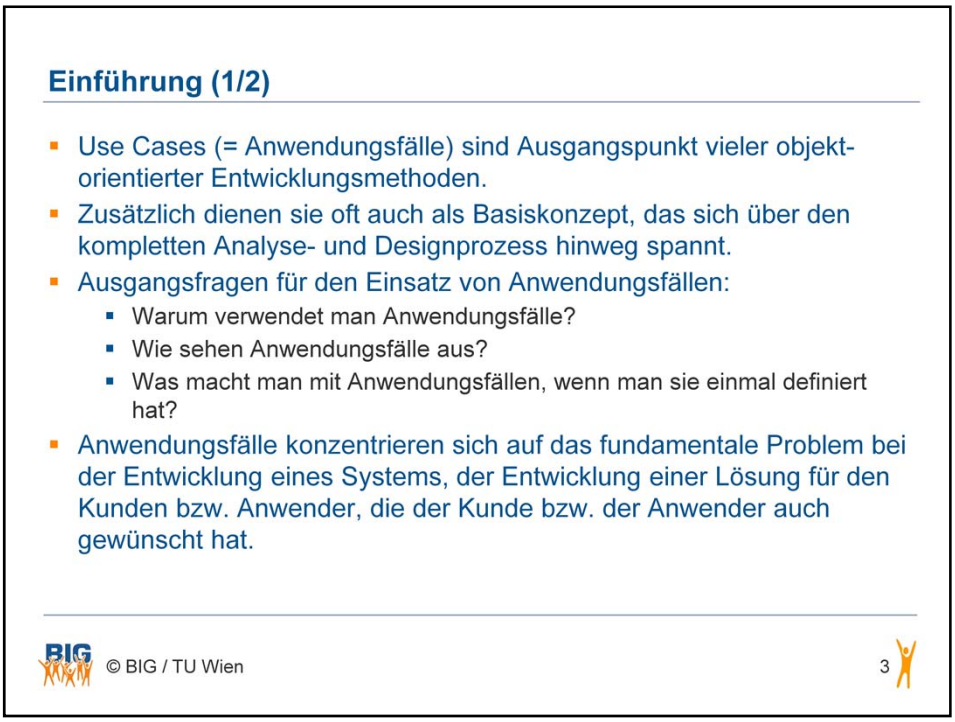
Juli 2012

ISBN 3898647765

- ***Anwendungsfalldiagramm***
- *Strukturmodellierung*
- *Zustandsdiagramm*
- *Sequenzdiagramm*
- *Aktivitätsdiagramm*



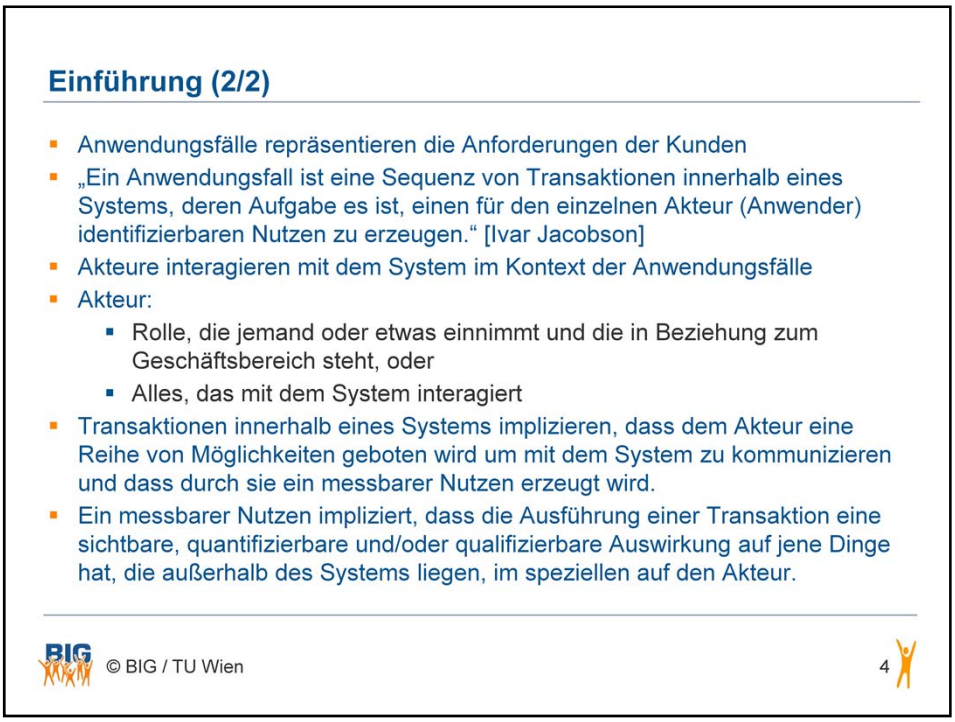
Was machen wir heute alles durch? Zuerst gibt es eine kurze **Einleitung** mit den wichtigsten Fakten. Wir schauen uns dann die **Akteure**, die Strichmanderl an, dann die **Anwendungsfälle**, und dann haben wir noch drei wesentliche Arten von **Beziehungen**, nämlich Beziehungen zwischen Anwendungsfällen und Akteuren, Beziehungen zwischen Anwendungsfällen und Beziehungen zwischen Akteuren. Ganz wesentlich dabei ist etwas, das wir nicht intensiv durchmachen werden, aber in der Praxis eigentlich das Wesentlichste ist und zwar die **Anwendungsfallbeschreibung**, denn so eine Ellipse alleine sagt nicht viel aus, viel wichtiger ist es, den Anwendungsfall, den sie darstellt, zu beschreiben.



Was sind jetzt Anwendungsfälle? Als erstes steht hier: Anwendungsfälle sind **Ausgangspunkt vieler objektorientierter Entwicklungsmethoden**. Ich behaupte einmal, dass sie an sich nicht viel mit Objektorientierung zu tun haben. Sie wurden nur populär zum selben Zeitpunkt wie die objektorientierte Modellierung populär wurde und einer der Gründer der UML, Ivar Jacobson, hat die „Use Case Centric"-Entwicklungsmethode eben bei Ericsson erfolgreich eingeführt und daher sind Anwendungsfalldiagramme ein Werkzeug, das vor allem in objektorientierten Entwicklungsmethoden vorkommt. Aber man muss nicht objektorientiert entwickeln, um Use Cases sinnvoll einsetzen zu können.

Anwendungsfälle sind eher ein **Basiskonzept**, das sich durch Anforderungserhebung, Analyse, Design, Implementierung, Test und Wartung durchgängig durchzieht. Wenn ich heute in dieser Lehrveranstaltung von Anwendungsfällen spreche, dann geht es um das Anwendungsfalldiagramm wie es in der Anforderungserhebung eingesetzt wird. Was immer dort erhoben wird, sollte nachher im weiteren Softwareentwicklungsprozess genutzt werden. Ich gehe kurz nachher an bestimmten Stellen darauf ein, aber heute lernen wir das Anwendungsfalldiagramm so kennen, wie es in der Anforderungserhebung eingesetzt wird.

Warum verwendet man Anwendungsfälle? Wie sehen Anwendungsfälle denn typischer Weise aus? Und was mache ich damit, wenn ich sie einmal gefunden habe? Im Prinzip sagt man, dass sich Anwendungsfälle auf das **fundamentale Problem** bei der Entwicklung eines Systems für einen Kunden konzentrieren. Weil derjenige, der dafür zahlt, möchte eine gewisse Funktionalität implementiert haben. Und meistens ist es aber so, dass die Informatiker oft einfach auf gewissen Annahmen basierend etwas programmieren, weil es für sie sinnvoll erscheint. D.h. der Anwendungsfall sollte dazu dienen, die Anforderungen so zu erheben, dass sie in einfachen Worten beschrieben sind, sodass sie der Anwender versteht. Die Summe der Anwendungsfälle beschreibt, wie sich der Kunde sein System vorstellt, wie er als Anwender damit interagieren möchte um eine Lösung zu bekommen. Bei den Anwendungsfällen geht es jedoch nicht darum, wie das System dahinter funktioniert.



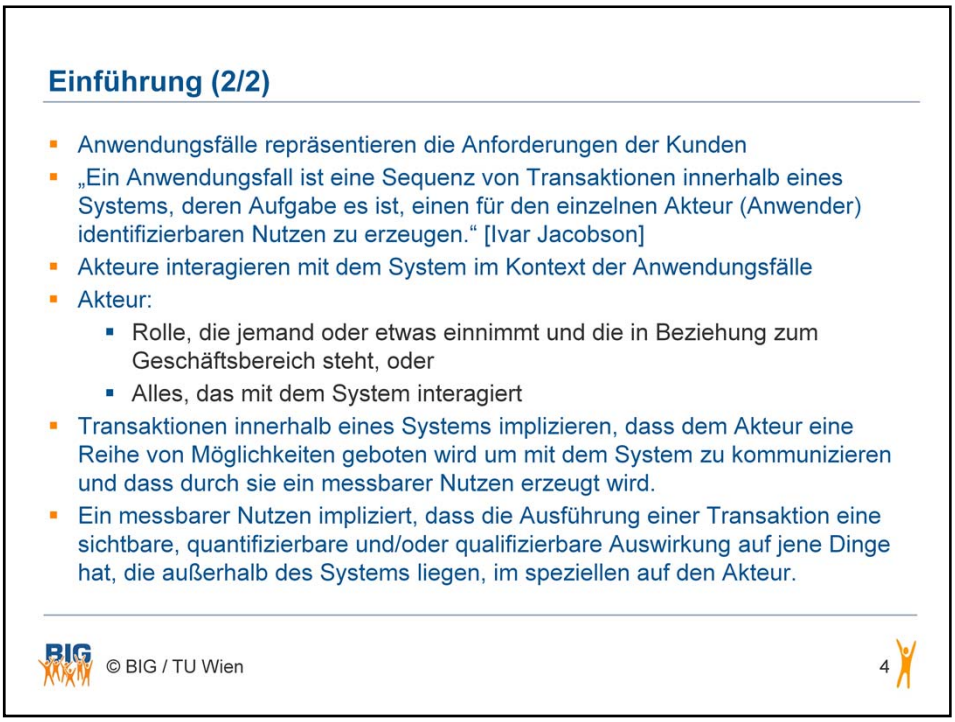
Anwendungsfälle repräsentieren **Anforderungen des Kunden**. Das ist jetzt leicht gesagt. Für mich stellt sich eher die Frage: Warum heißt das Ganze „Anwendungsfall“? Ein Anwendungsfall ist ein typischer Fall, den das System behandeln muss. Ich sollte mich daher fragen, welche Fälle denn alle auftreten könnten, anstatt nur in reinen Systemfunktionalitäten zu denken.

Ich probiere das nun an Beispielen zu beschreiben.

Einen Fall, den wir früher in unserer Lehrveranstaltung OOM hatten, war, dass alle Studenten für die Übungen schon am Montag in der Früh kreuzen mussten, nicht nur jene, die am Montag Übung hatten, sondern auch jene, die am Freitag Übung hatten. Hintergrund war, dass es nur einen Zeitpunkt gab, an dem Studenten kreuzen konnten. Wir mussten also den frühest möglichen Zeitpunkt nehmen. In Anwendungsfällen gedacht, müsste man so vorgehen: Es kann doch der Fall auftreten, dass es nur eine einzige VU gibt, jedoch zu dieser VU unterschiedliche Gruppen abgehalten werden und diese Gruppen finden an unterschiedlichen Terminen statt. Ich möchte es nun ermöglichen, dass für unterschiedliche Termine unterschiedlich gekreuzt werden kann. Das wäre ein Anwendungsfall.

Noch ein Beispiel aus unserer Lehrveranstaltung OOM. Wir haben im Wintersemester 5 – 6 und im Sommersemester 10 – 12 Übungsgruppen. Prinzipiell wäre es mir egal, wann Sie in welcher Woche in welcher Übungsgruppe erscheinen. Trotzdem haben wir aber am Beginn vom Semester ausgegeben, dass Sie immer in dieselbe Übungsgruppe kommen müssen und nicht die Gruppen tauschen dürfen. Das haben wir ganz am Anfang nicht gehabt. Die Konsequenz war, dass ca. 20% der Studierenden immer in eine andere Übungsgruppe gegangen sind. Dann stehe ich mit meinen Listen, wo die die Kreuzerl vermerkt sind, da und muss mir die anwesenden Studierenden auf unterschiedlichen Listen zusammensuchen. Das ist nicht administrierbar. Wenn ich nun in Anwendungsfällen denke, könnte es doch passieren, dass ein Student, aus welchem Grund auch immer, für eine Übungsgruppe verhindert ist, und in eine andere Übungsgruppe gehen möchte. Das wäre ein typischer Anwendungsfall und ich müsste mir überlegen, wie die Teilnehmer am System mit diesem Anwendungsfall umgehen. In diesem Anwendungsfall müsste jeder Woche für Woche bekannt geben, dass er in eine andere Gruppe wechseln möchte und dann werden für den Lehrveranstaltungsleiter die Listen mit den Kreuzerln entsprechend angepasst. Die Anwendungsfallbeschreibung sollte es mir dann auch ermöglichen, die Kosten für die Implementierung des Anwendungsfalls abzuschätzen. Dann kann ich Kosten und Nutzen gegenüberstellen. Für den eben geschilderten Fall war anscheinend der Nutzen nicht hoch genug, um es im System tatsächlich zu implementieren.

Noch ein Beispiel aus dem Bereich der Restaurants: Es könnte der Fall auftreten, dass sich die Gäste die Rechnung teilen möchten. D.h. Sie möchten den Rechnungsbetrag durch die Anzahl der Gäste dividieren und jeder gleich viel zahlen. Da könnte das Problem entstehen, dass Systeme bei Kreditkartenrechnungen nur einen Betrag abrechnen können, der auch auf der Rechnung steht. Dann habe ich sozusagen ein Requirement nicht erfüllt, denn es kommt eben durchaus vor, der Anwendungsfall, dass Gäste sich den Betrag teilen wollen. Beim Anwendungsfalldiagramm geht es genau darum, alle diese Fälle im Vorhinein zu erheben. Oft sind es – leider sag ich jetzt einmal teilweise auch auf unseren Folien – immer die trivialen Dinge, die man sieht. In einem Studentensystem habe ich dann die Trivialfälle: Einen neuen Studenten anlegen, einen Studenten exmatrikulieren, zu einer Lehrveranstaltung anmelden usw.. Hier handelt es sich natürlich auch um Anforderungen und Anwendungsfälle. Aber wichtig im anwendungsfallzentrierten Denken und der Grund dafür, warum dass es so erfolgreich eingesetzt werden kann, ist der Gedanke, sich im Vorhinein zu überlegen, was denn wirklich alles passieren kann. Und darum sage ich auch immer, dass das Schwierigste ist, sich etwas für die Erstversion eines Systems zu überlegen. Weil wenn ich die Zweitversion von meinem System entwickle, dann habe ich die Erstversion ja schon im Einsatz gehabt, dann weiß ich, was alles passieren kann und was alles nicht funktioniert. Die Kunst ist es sozusagen, das schon für die Erstversion vorzuziehen. Bei der Zweitversion ist, muss man sagen, eher das Problem, dass man unter „Feature-itis“ leidet, wenn man weiß, was alles nicht funktioniert hat, dann will man vielleicht zu viel an Anwendungsfällen in das System packen. Das war ein kurzer Ausflug im Rahmen der Frage: Was sind Anwendungsfälle und was heißt „Fall“ in diesem Zusammenhang.



Weiter auf den Folien:

Ein **Anwendungsfall** ist auch definiert als eine **Sequenz von Transaktionen** innerhalb eines Systems, deren Aufgabe es ist, einen für einen einzelnen Akteur (Anwender) identifizierbaren Nutzen zu erzeugen. Das klingt super, spektakulär, richtig akademisch. Aber was steckt dahinter? Es geht nicht um Transaktionen im Sinne von Datenbanken. Sie können es eigentlich aus meiner Sicht ein bisschen auch mit „Schritte“ übersetzen. Was will ich damit sagen? Schreiben Sie eine Bedienungsanleitung über Ihren DVD Rekorder. Auch mit dem werden Sie einen Use Case haben: „Eine neue Sendung aufnehmen“. Dann gibt es da gewisse Schritte, die Sie durchführen müssen, Sie als „User“. Es wird beschrieben, wie Sie als User mit dem System – in diesem Fall ist es ja eher eine Hardware mit ein paar Knöpfen – interagieren. Und das ist unter „Sequenz von Transaktionen“ zu verstehen. Dieses Interagieren von Ihnen als User mit dem System DVD Rekorder.

Akteure interagieren mit dem System im Kontext der Anwendungsfälle, d.h. alles was ich vornehmen kann, ist ein Anwendungsfall. Man könnte auch sagen, dass das Benutzerhandbuch die Summe der Anwendungsfälle ist, und das deckt hoffentlich alles ab, was an Funktionalität gewünscht ist. Ein „Akteur“ ist eine Rolle, die jemand – und ganz wichtig – oder etwas einnimmt, und in Beziehung mit dem Geschäftsbereich steht. Z.B. bin ich nicht Teil des DVD-Rekorders, der DVD-Rekorder ist das System und ich als Akteur bediene es. Das ist ganz trivial, wenn man über so Dinge wie einen DVD-Rekorder redet. Wenn Sie hier jetzt ein System entwickeln für – ich sag jetzt einmal - eine Arztpraxis, stellet sich die Frage: Ist der Patient, ein Akteur dieses Systems? Beide Antworten „ja“ und „nein“ sind richtig. Denn es kommt darauf an, was ich unter dem System verstehe. Nämlich, verstehe ich darunter die IT-Anwendung oder verstehe ich das Sozio-Ökonomische-System darunter? Betrachten wir dazu einen Anwendungsfall „Patientendaten aufnehmen“. Natürlich wird der Patient seine Daten bekannt geben. Im Sozio-Ökonomischen-System ist der Patient daher ein Akteur. Jedoch wenn ich nur auf das IT-System blicke, dann wird der Patient kaum das IT-System der Arztpraxis bedienen. D.h. es wird eine Arztpraxis Angestellte, eine Sekretariats-Kraft, eventuell hier die Patientendaten im System pflegen. Im Sinne des IT-Systems ist der Patient kein Akteur, weil er nicht mit dem System er Arztpraxis interagiert.

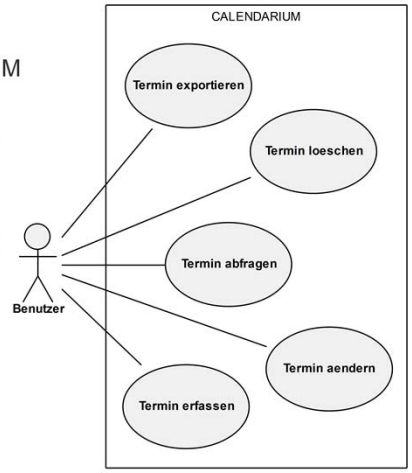
Transaktionen innerhalb eines Systems implizieren, dass dem Akteur eine Reihe von Möglichkeiten geboten werden, um mit dem System zu kommunizieren und dass dadurch ein messbarer Nutzen erzeugt wird.

Der **messbare Nutzen** ist von großer Bedeutung. Überlegen wir uns das an einem Beispiel: Sie kaufen sich im Online-Buchhandel ein Buch. dann ist wahrscheinlich das Kaufen des Buches ein Anwendungsfall, das Eingeben der Kreditkartennummer ist kein Anwendungsfall, weil kein Mensch will eine Kreditkartennummer eingeben, das erzeugt keinen Nutzen – das Buch zu kaufen erzeugt den Nutzen. Obwohl so trivial ist das nicht immer, weil ich muss schon auch bedenken, dass es den Bedarf geben könnte, dass Anwender, Kunden, häufig umziehen und alle drei Jahre eine neue Kreditkarte bekommen und dann generiert das Verwalten von Stammdaten natürlich schon einen Nutzen für den Online-Buchhändler. Es ist also nicht immer so einfach zu trennen, was einen messbaren Nutzen mit sich bringt und was nicht. Aber wenn ich einmalig einkaufe, ohne dass ich mich als Kunde beim Online-Buchhändler registriere, dann ist das Eingeben der Kreditkartennummer auf keinen Fall ein Anwendungsfall. Weil damit für niemanden ein Nutzen verbunden ist.

Ich möchte Ihnen nun anhand eines sehr einfachen Systems – einem Kaugummiautomaten – beschreiben, wie man die Interaktionen des Benutzers mit dem System festlegt, in anderen Worten, wie man den Standardablauf der Benutzerinteraktionen beschreibt. Der Kaugummiautomat hat nur zwei Anwendungsfälle und das sind „Kaugummi kaufen“ und wahrscheinlich „Kaugummis nachfüllen“ für den Betreiber. Also wir betrachten „Kaugummi kaufen“ als Anwendungsfall. Wie der Nutzer damit interagiert, das kriegt bereits ein kleines Kind nach kurzer Zeit relativ schön heraus, nämlich, es wirft die Münze ein, es dreht den Hebel nach rechts, es macht die Klappe auf und nimmt sich die Kaugummis heraus. Das ist eine Menge von Schritten, ein Set von Transaktionen. Das ist das, was wir nachher als „Standardablauf“ bezeichnen. Wenn Sie die Anwendungsfallbeschreibung weiter hinten in den Folien sehen, so sind es in diesem Fall genau diese 4 Schritte, die sie bei „Standardablauf“ beschreiben sollten. Warum hebe ich das jetzt so hervor? Sie denken sich vielleicht, dass das eh klar ist. Weil ich kein Maschinenbauer bin, wäre ich nie auf die Idee gekommen zu fragen, warum da jetzt Kaugummis herausfallen, wie der Mechanismus funktioniert, dass sich irgendetwas öffnet und die Kaugummis herausfallen. Genauso muss ich aber bei der Anwendungsfallbeschreibungen denken. Das ist wie beim Auto fahren, das war früher ein Beispiel in unseren Folien: Ich weiß als Autofahrer, als Benutzer, wenn ich auf das Gas steige und ich habe den Vorwärtsgang drinnen, dann fährt das Auto nach vorne. Ich habe aber überhaupt keine Ahnung, wie der Tropfen Benzin das letztendlich alles antreibt. Warum streiche ich das so heraus? Weil meistens wird der Anwendungsfall eingesetzt im Software Engineering und Informatiker sind damit betraut und die haben einen Nachteil – Sie wollen immer das System dahinter beschreiben, wie es funktioniert und was abläuft. Das ist jedoch nicht Aufgabe des Anwendungsfalls oder der Anwendungsfallbeschreibung in der Anforderungserhebung. Großteils passieren hier die Fehler, weil in der Anwendungsfallbeschreibung schon das System beschrieben wird. Darum geht es aber bei der Anforderungserhebung nicht. Ich soll nur beschreiben, wie der Benutzer mit dem System interagiert: Münze einwerfen, Hebel drehen, Klappe öffnen, Kaugummi entnehmen. Wie der Automat dahinter funktioniert, ist in der Anforderungsbeschreibung noch unerheblich. Ich habe ja am Anfang der Einheit festgehalten, dass ich mich hier in dieser Einheit auf die Anforderungserhebung mittels Anwendungsfalldiagrammen beziehe. Der Anwendungsfall sollte sich später durch den ganzen Software Engineering Prozess spannen. Sie sollten sich dann schon bei der Analyse und dem Design fragen, wie denn das ganze realisiert wird, ob ich denn auch das erreiche, was ich in den Anforderungen festgehalten habe. Aber in der Anforderungserhebung sollten Sie nicht aufnehmen, wie das System realisiert wird. Angenommen, sie müssen die Anwendungsfallerhebung für unser E-Learning System TUWEL machen. Aus dem einen Grund sind Sie sehr gut dafür geeignet und aus dem anderen Grund sind Sie fast schlecht dafür geeignet. Warum Sie gut geeignet sind, ist, weil Sie Studenten sind und wissen, was Sie von so einem System gerne hätten, warum Sie schlecht geeignet sind ist, weil Sie Informatiker sind und immer sofort an die Lösung denken und nicht an die Anforderungen. Viele der Mails, die wir an oom@big.tuwien.ac.at bekommen, bzw. Forum-Postings, machen Vorschläge zur System-Verbesserung. Das wären Kandidaten für Anwendungsfälle für die Fortentwicklung unseres TUWEL-Systems.

Bsp.: CALENDARIUM

- **System** (was wird beschrieben?)
  - der Onlinekalender CALENDARIUM
- **Akteur** (wer benutzt das System?)
  - Benutzer des Kalenders
- **Anwendungsfälle des Benutzers** (was machen die Akteure?)
  - Abfragen von Terminen
  - Exportieren von Terminen
  - Löschen von Terminen
  - Ändern von Terminen
  - Erfassen von Terminen



Hier sehen wir schon ein kleines Beispiel anhand unseres CALENDARIUMs, bevor wir noch auf die Einzelteile des Anwendungsfalldiagramms genauer eingehen. Das Strichmännchen ist der Akteur, die Use Cases oder Anwendungsfälle sind die Ellipsen und die Anwendungsfälle werden umgeben von einer Systemgrenze.

Akteur

- Akteure **interagieren** mit dem System…
  - indem sie das **System benutzen**,
    d.h. die Ausführung von Anwendungsfällen initiieren
  - indem sie **vom System benutzt werden**,
    d.h. Funktionalität zur Realisierung von Anwendungsfällen zur Verfügung stellen
- Akteur wird **durch Assoziationen mit Anwendungsfällen verbunden,** d.h. er »kommuniziert« mit dem System
- Jeder Akteur muss mit **mindestens einem Anwendungsfall** kommunizieren
- Die Assoziation ist binär und kann **Multiplizitäten** aufweisen
- Notationsvarianten:

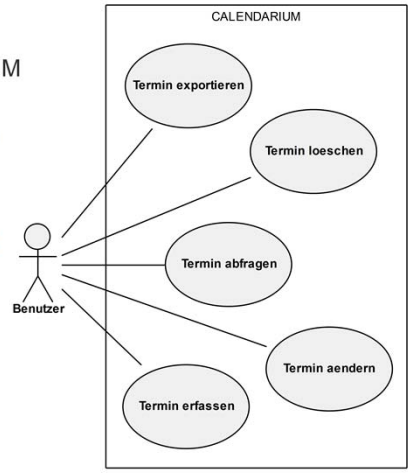
7

Ein **Akteur** ist jemand, der außerhalb des Systems steht und mit dem System **interagiert**. Die Akteure werden durch **Assoziationen** mit den Anwendungsfällen verbunden.

Schauen wir uns das noch einmal am Beispiel auf der vorhergehenden Folie an.

Bsp.: CALENDARIUM

- **System** (was wird beschrieben?)
  - der Onlinekalender CALENDARIUM
- **Akteur** (wer benutzt das System?)
  - Benutzer des Kalenders
- **Anwendungsfälle des Benutzers** (was machen die Akteure?)
  - Abfragen von Terminen
  - Exportieren von Terminen
  - Löschen von Terminen
  - Änderung von Terminen
  - Erfassung von Terminen



An diesem Beispiel sehen wir, dass der Akteur „Benutzer“ mittels Beziehungen, Assoziationen, mit den Anwendungsfällen verbunden ist und auch außerhalb der Systemgrenzen steht. In diesem Beispiel bedeutet es, dass der Benutzer „Termin abfragen“, „Termine exportieren“ usw. ausführen kann. Das ist in diesem Beispiel trivial, weil der Benutzer der einzige Akteur ist und hier mit allen Anwendungsfällen verbunden ist. Aber nur wenn eine Beziehung besteht, kann er den Anwendungsfall auch tatsächlich ausführen. Ausnahmen sind später Generalisierungen und im Best Practice Fall auch include- und extends-Beziehungen, aber darauf werden wir später noch eingehen.

Akteur

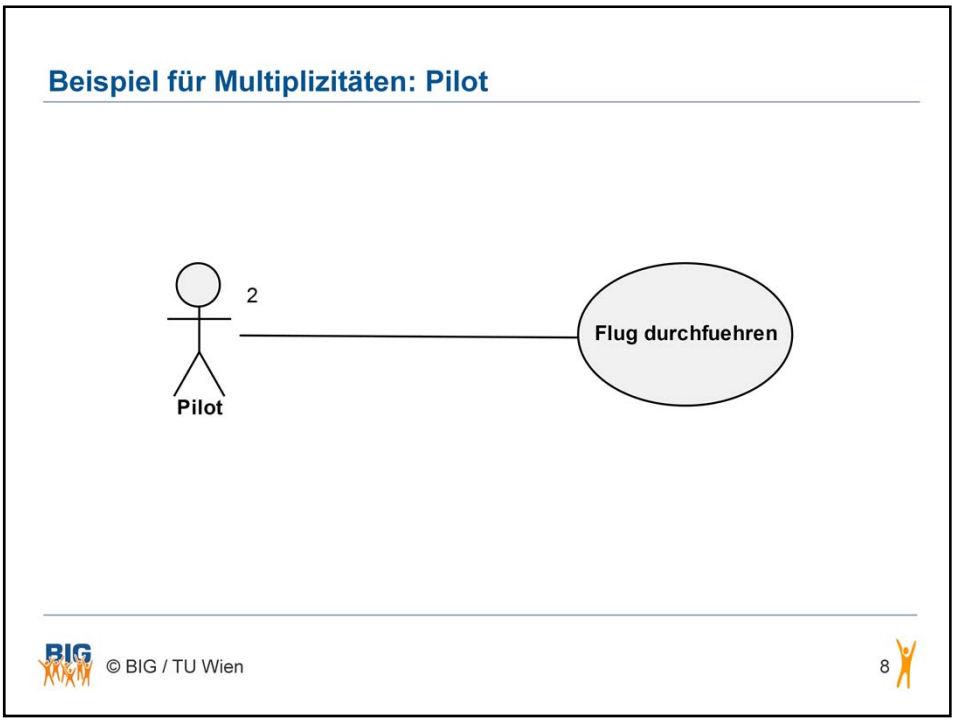
- Akteure **interagieren** mit dem System…
  - indem sie das **System benutzen**,
    d.h. die Ausführung von Anwendungsfällen initiieren
  - indem sie **vom System benutzt werden**,
    d.h. Funktionalität zur Realisierung von Anwendungsfällen zur Verfügung stellen
- Akteur wird **durch Assoziationen mit Anwendungsfällen verbunden,** d.h. er »kommuniziert« mit dem System
- Jeder Akteur muss mit **mindestens einem Anwendungsfall** kommunizieren
- Die Assoziation ist binär und kann **Multiplizitäten** aufweisen
- Notationsvarianten:

7

Normalerweise **benutzen** die Akteure das System. Sie können aber auch vom System **benutzt werden**. Wann ist das der Fall? – Wenn wir an menschliche User denken, dann ist es meistens so, dass Sie das System benutzen. Sehr oft ist es bei nicht-menschlichen Akteuren der Fall, dass sie vom System benutzt werden, denn ein anderes System, das ich nur benutzen und nicht weiterentwickeln möchte, könnte auch ein Akteur sein. Nehmen wir als Beispiel wieder TUWEL. Angenommen wir möchten für TUWEL eine Anforderungsanalyse mittels Anwendungsfalldiagrammen machen. Wir wissen aber, dass es ein bestehendes System TISS gibt, wo letztendlich die ganze Zeugnisverwaltung usw. funktioniert. Es wäre jetzt wünschenswert, wenn ich die Funktionen von TISS aus dem TUWEL heraus anstoßen könnte und die Daten von TISS übernommen werden. TUWEL verändert TISS nicht, es nutzt nur TISS. Dementsprechend ist das TISS aus der Sicht des TUWELs einfach ein Akteur. TISS ist ein Akteur, der jetzt vom System TUWEL, das ich entwickeln möchte, benutzt wird.

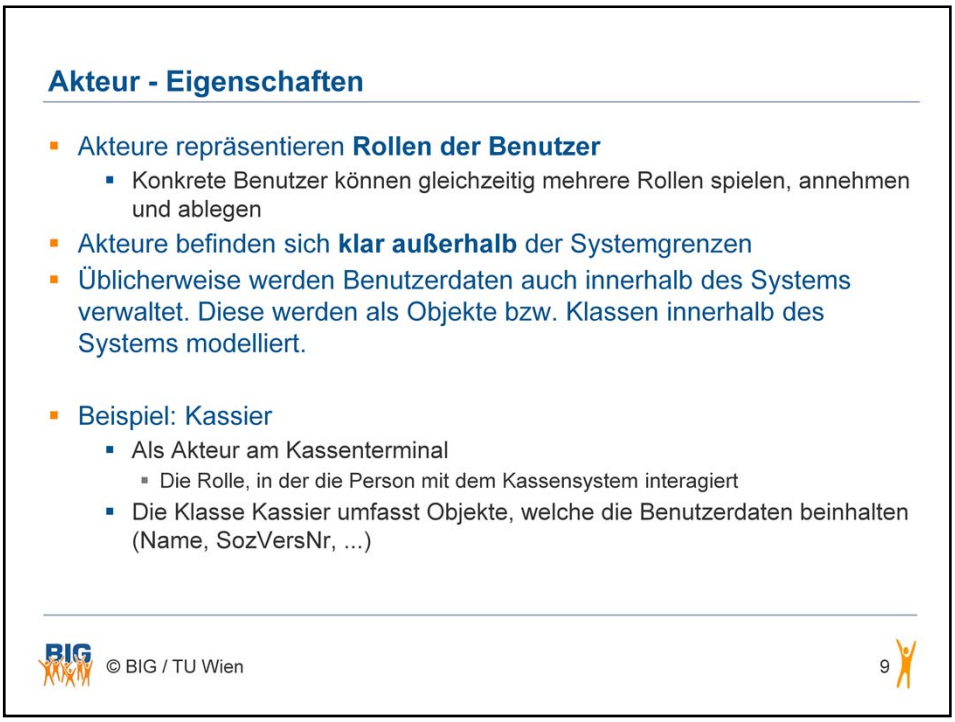
Jeder Akteur **muss mit mindestens einem Anwendungsfall kommunizieren**. Das ist trivial, weil wenn er mit keinem einzigen Anwendungsfall interagieren kann, dann kann er auch nicht mit dem System interagieren, und dann brauchen wir ihn nicht als Benutzer für unser System zu berücksichtigen.

Die Beziehung zwischen Akteuren und Anwendungsfällen ist **binär**, was dadurch schon festgelegt ist (immer von einem Benutzer zu einem Anwendungsfall), und sie kann **Multiplizitäten** aufweisen. Dies sehen wir auf der nächsten Folie.



Hier sehen wir die Verwendung von Multiplizitäten an einem Beispiel.

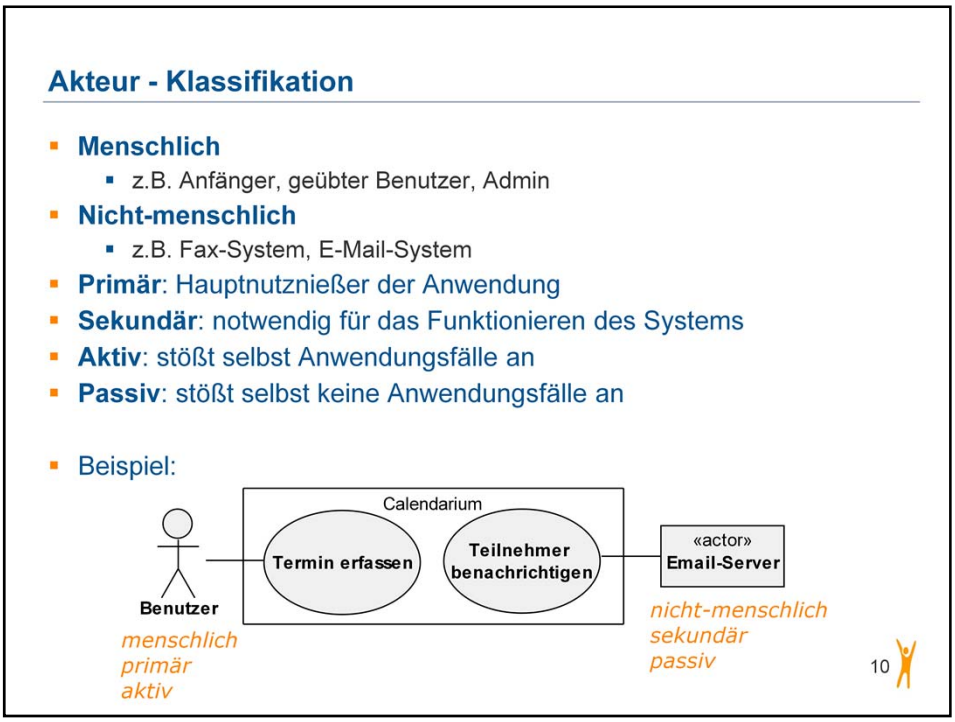
Wir haben hier den Akteur „Pilot“ und den Anwendungsfall „Flug durchführen“. Ich kann nun mit Multiplizitäten ausdrücken, dass „Flug durchführen“ von zwei Piloten durchgeführt wird. Ganz interessant, aber nur für den Akademiker oder Theoretiker relevant, ist der umgekehrte Fall: Wenn wir keine Multiplizität angeben, nehmen wir meistens bei graphischen Notationen 1 an. Das Anwendungsfalldiagramm stellt hier eine Ausnahme dar und zwar unterstellen wir implizit auf der Seite des Anwendungsfalls, auch wenn wir es nicht immer dazu schreiben, dass da eigentlich ein Stern stehen müsste, obwohl ich das in der Praxis noch nie irgendwo gesehen habe. Denn hoffentlich kann ein Pilot den Use Case „Flug durchführen“ öfters als nur einmal durchführen. Auf die Seite des Anwendungsfalls wird in der Praxis nie eine Multiplizität angegeben und man nimmt immer hier an, dass der Use Case öfters durchgeführt werden kann. Die Multiplizitäten zeichnet man also praktisch immer nur auf der Seite der Akteure ein.



Akteure repräsentieren **Rollen der Benutzer**, also Sie modellieren nicht den Christian Huemer als Akteur, sondern Sie können ihn im System z.B. als „Lehrenden“ betrachten. Wenn ich jetzt zusätzlich noch als Christian Huemer studieren würde, dann wäre das auch nicht tragisch, weil in manchen Anwendungsfällen kann ich den Akteur „Student“ übernehmen und in anderen kann ich den Akteur „Lehrender“ übernehmen. Sie müssen nicht erheben, dass rein theoretisch ein „Lehrender“ auch noch ein „Studierender“ sein kann, das ist eher unerheblich.

Üblicherweise werden auch die **Benutzerdaten** im System verwaltet, aber da muss man wohl unterscheiden. Akteure stehen außerhalb des Systems, ihre Benutzerdaten werden innerhalb des Systems mit Hilfe von Klassendiagrammen modelliert.

Hier ein Beispiel: Der Kassier am Kassenterminal ist ein Akteur und steht außerhalb der Systemgrenzen. Er interagiert mit der Kassa. Jedoch die Benutzerdaten des Kassiers werden als Klasse „Kassier“ im Klassendiagramm erhoben.



Akteure kann ich **klassifizieren**.

Wir klassifizieren in **menschliche** und **nicht-menschliche** Akteure. So wie ich es vorhin beschrieben habe, ist TISS ein nicht-menschlicher Akteur, wenn das zu entwickelnde TUWEL darauf zugreift. Wenn ich als „Lehrender“ TUWEL bediene, dann bin ich ein menschlicher Akteur.

Unten sieht man als Beispiel einen Benutzer, der ist als Strichmanderl dargestellt und dann sieht man einen E-Mail-Server, der ist als Viereck dargestellt und in den Spitzklammern wird Akteur, also <<actor>> in englisch, hineingeschrieben. Das legt vielleicht die Vermutung nahe, dass alles was menschlich ist, als Strichmanderl dargestellt wird und alles was nicht menschlich ist, wie rechts dargestellt notiert wird. Das stimmt nicht. Das sind zwei äquivalente Symbole. Sie könnten sie sogar umdrehen, d.h. Sie könnten den E-Mail-Server mit einem Strichmanderl darstellen und Sie könnten den Benutzer mit dem eckigen Kasterl notieren. Ganz allgemein gilt für die UML immer die Notation, die Sie rechts sehen, nämlich ein Viereck in dem in Spitzklammern die Metaklasse angegeben ist. Diese Notation ist immer erlaubt. D.h. Sie könnten auch anstatt, dass Sie Aktivitäten in Kästchen mit abgerundeten Kanten schreiben, Viereck verwenden und in Spitzklammern <<activity>> angeben. D.h. die Syntax ist hier nicht entscheidend für die Unterscheidung zwischen menschlichen und nicht-menschlichen Akteuren.

Wir unterscheiden desweiteren zwischen **primären** und **sekundären** Akteuren. Der primäre Akteur ist der Hauptnutzneißer der Anwendung und der sekundäre ist für die Anwendung auch notwendig. Angenommen wir machen ein Prüfungsgespräch. Der Hauptnutznießer sind Sie, damit sind Sie der primäre Akteur und ich muss auch noch dabei sein als Lehrender, dann wäre ich sekundär.

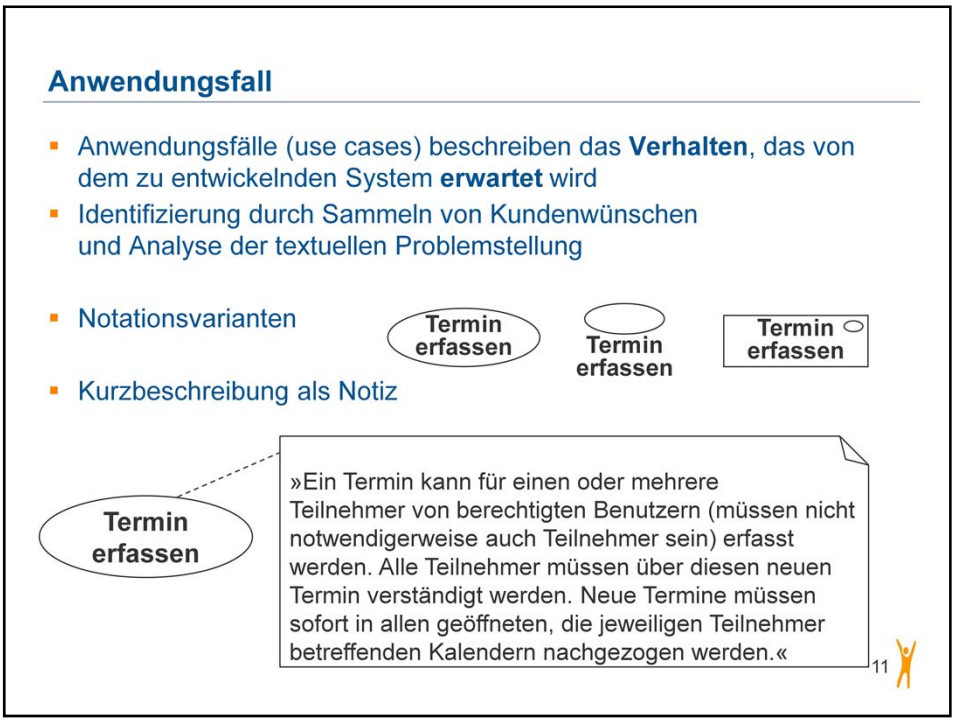
Wir können auch zwischen **aktiv** und **passiv** unterscheiden. Wenn der unten dargestellte E-Mail-Server beispielsweise angestoßen, d.h. benutzt wird, dann wäre er ein passiver Akteur. Oder wenn ich also „Lehrender“ die Benotung durchführe, so stoße ich eher aktiv den Prozess an, dass die Daten an TISS übergeben werden. TISS ist dann am Anwendungsfall beteiligt, aber passiv.

Jetzt kommt eine schwierige Frage, wie sich immer bei der Übung herausgestellt hat. Nämlich sind die Klassifikationen, d.h. die erste Klassifikation menschlich/nicht-menschlich, die zweite primär/sekundär und die dritte aktiv/passiv **Merkmale des Akteurs oder Merkmale der Beziehungen** zwischen dem Anwendungsfall und dem Akteur?

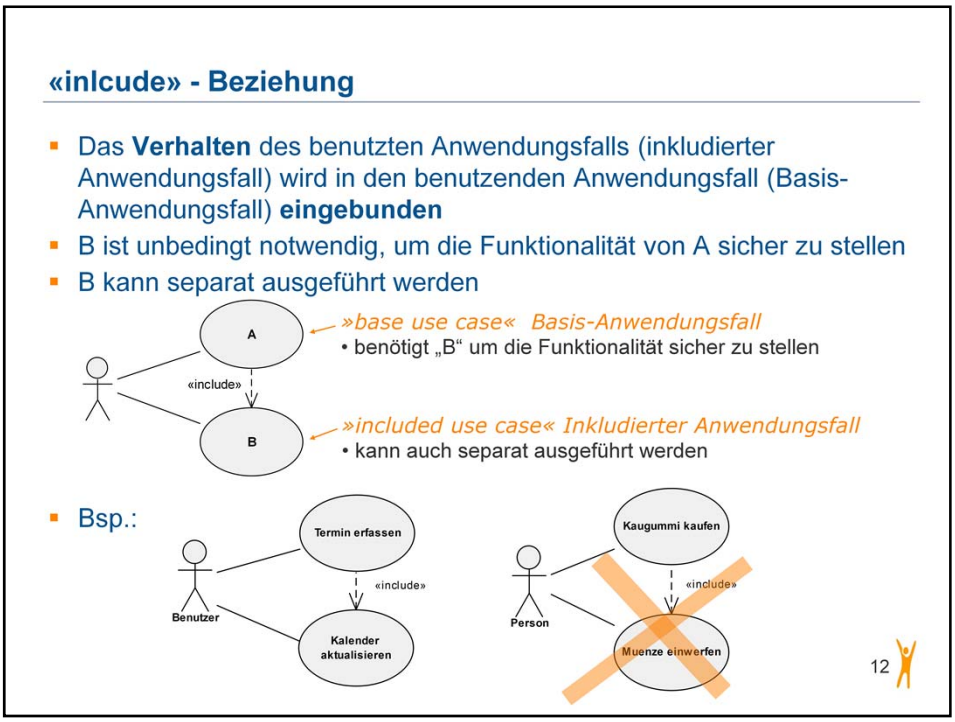
Gehen wir es von oben durch. Ist die Unterscheidung menschlich/nicht-menschlich ein Merkmal von einem Akteur oder von der Beziehung? – Vom Akteur, weil ich bin immer menschlich, unabhängig vom Anwendungsfall.

primär/sekundär ist eigentlich ein Merkmal der Beziehung. Ich kann beispielsweise in einem Fall mit dem System „prüfen“, dann bin ich wie vorhin gesagt der sekundäre Akteur. Bei der Prüfungsgeldabrechnung oder auch bei anderen Funktionen im System bin ich allerdings der primäre Akteur. D.h. es ist nicht vom Akteur abhängig, der kann bei einem Anwendungsfall primär oder beim anderen sekundär sein, sondern es ist ein Merkmal der Beziehung zwischen Akteur und Anwendungsfall.

Das gleiche gilt bei aktiv/passiv. Ich kann den einen Anwendungsfall anstoßen, den anderen Anwendungsfall stoße ich dann aber nicht an. Daher wäre das ein Merkmal der Beziehung zwischen einem Akteur und einem Anwendungsfall.



Den **Anwendungsfall**, habe ich selbst jetzt schon oft vorgestellt. Nun kommen wir zu seiner Notation. Er wird als Ellipse dargestellt und bei der Darstellungsform gibt es je nach Tool Unterschiede: Die einen schreiben den Namen des Anwendungsfalls in die Ellipse, die anderen schreiben ihn darunter, die nächsten verwenden die Viereckknotation wo in der rechten oberen Ecke des Vierecks eine kleine Ellipse eingezeichnet ist. Man kann wie bei allen Modellelementen auch noch eine Notiz dazu fügen. Das würde ich allerdings nie machen, weil damit Sie einen Anwendungsfall vernünftig beschreiben, benötigen Sie ein bis zwei Seiten, und die werden Sie nicht als Notiz dazu hängen.



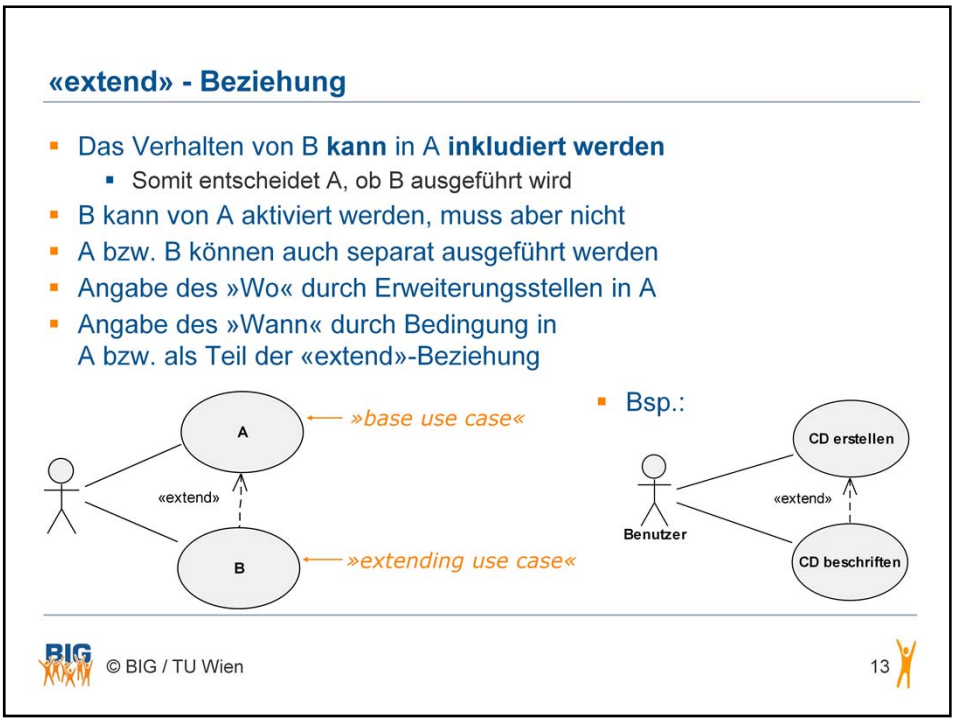
Wir kommen zu den Beziehungen zwischen Anwendungsfällen. Die erste Beziehung die wir durchnehmen ist die **include**-Beziehung. Die include-Beziehung bedeutet, dass wenn eine Instanz des Basis Anwendungsfalles – in unserem Falle hier „A“ – ausgeführt wird, dann muss immer auch eine Instanz des inkludierten Anwendungsfalles „B“ ausgeführt werden.

Ich zeige hier kurz einen häufigen Fehler an unserem Automaten-Beispiel. Ich könnte beim Kaugummiautomaten sagen, ich habe den Anwendungsfall „Kaugummi kaufen“ und der inkludiert den Anwendungsfall „Münze einwerfen“. Kling doch logisch. Stimmt das aufgrund dessen, was ich vorher gesagt habe? – Nein. Warum nein? – Weil „Münze einwerfen“ keinen Nutzen darstellt und damit auch kein Anwendungsfall ist.

Die include-Beziehung deutet immer darauf hin, dass wenn „A“ durchgeführt wird, auch „B“ durchgeführt wird, aber was trotzdem noch gegeben sein muss ist, dass „B“ alleine auch durchgeführt werden kann und „Münze einwerfen“ kann alleine nicht ausgeführt werden, denn das erzeugt keinen Nutzen, eher einen Verlust. Daher ist „Münze einwerfen“ kein Anwendungsfall. Daher gilt folgendes: Der inkludierte Anwendungsfall kann alleine ausgeführt werden, der Basis Anwendungsfall kann nicht alleine ausgeführt werden, er muss in Kombination mit dem inkludierten Anwendungsfall ausgeführt werden.

In dem andere Beispiel haben wir „Kalender aktualisieren“. Diesen Anwendungsfall könnte ich auch alleine ausführen. „Termin erfassen“ hingegen bedeutet, dass einerseits die Terminerfassung durchgeführt wird und andererseits im Zuge dessen auch „Kalender aktualisieren“ als Teil davon ausgeführt wird.

Noch ein Hinweis für die Übungen: Wenn Sie Beziehungen zwischen Anwendungsfällen wie eben die include-Beziehung oder die darauffolgende extends-Beziehung und die Generalisierung beschreiben müssen, zeichnen Sie bitte immer auch Akteure ein. Dies ist eine unbedingte Voraussetzung.

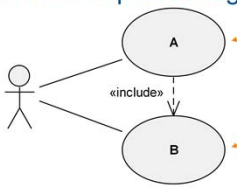


Dann gibt es noch die **extend**s-Beziehung zwischen Anwendungsfällen. Sie ist ähnlich zur include-Beziehung. Auch hier haben wir einen Basis Anwendungsfall und dazu kommt einen erweiternden Anwendungsfall, oder extending Use Case im Englischen. Das bedeutet, dass es unter Umständen sein könnte, dass wenn der Basis-Anwendungsfall „A“ durchgeführt wird, auch der erweiternde Anwendungsfall „B“ durchgeführt wird. Hier ist es jedoch keine Pflicht, dass „B“ durchgeführt wird, wenn „A“ ausgeführt wird. Beim include vorher haben wir gesagt, dass immer wenn der Basis Use Case „A“ durchgeführt wird, auch „B“ durchgeführt wird und beim extend ist es jetzt so, dass wenn „A“ durchgeführt wird, es nur sein könnte, dass „B“ durchgeführt wird.

Blöderweise richtet sich der Pfeil nach der natürlichen Sprache – nämlich nach dem include bzw. extends und die gehen vom logischen Satzbau, der Grammatik her, in die umgekehrte Richtung. Gehen wir dazu noch einmal zurück zur vorhergehenden Folie.

«inlcude» - Beziehung

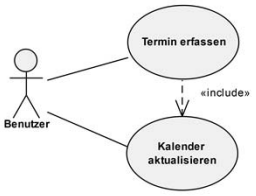
- Das **Verhalten** des benutzten Anwendungsfalls (inkludierter Anwendungsfall) wird in den benutzenden Anwendungsfall (Basis-Anwendungsfall) **eingebunden**
- B ist unbedingt notwendig, um die Funktionalität von A sicher zu stellen
- B kann separat ausgeführt werden

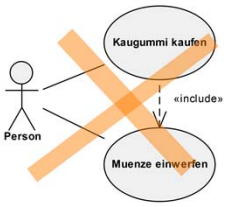


»base use case« Basis-Anwendungsfall
• benötigt „B“ um die Funktionalität sicher zu stellen

»included use case« Inkludierter Anwendungsfall
• kann auch separat ausgeführt werden

- Bsp.:



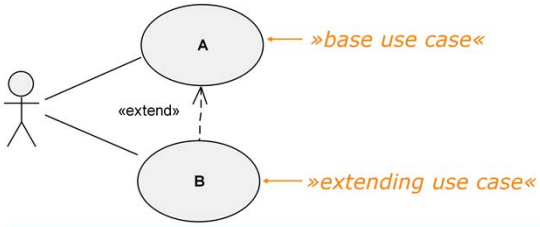


12

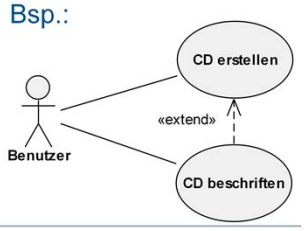
Bei der include-Beziehung sage wir folgendes: Der Basis Use Case „A“ inkludiert den inkludierten Use Case „B“, d.h. der Pfeil geht vom Basis Use Case zum inkludierten Use Case.

«extend» - Beziehung

- Das Verhalten von B **kann** in A **inkludiert werden**
  - Somit entscheidet A, ob B ausgeführt wird
- B kann von A aktiviert werden, muss aber nicht
- A bzw. B können auch separat ausgeführt werden
- Angabe des »Wo« durch Erweiterungsstellen in A
- Angabe des »Wann« durch Bedingung in A bzw. als Teil der «extend»-Beziehung



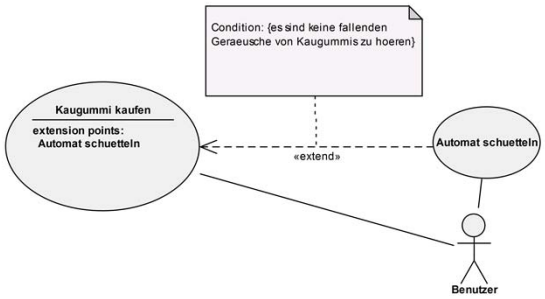
- Bsp.:



Und bei der extends-Beziehung ist es genau umgekehrt, weil es sich nach der natürlichen Sprache richtet. Der erweiternde Use Case „B“ erweitert den Basis Use Case „A“ und daher geht der Pfeil zum Basis Use Case. Das ist etwas, das auch sehr oft falsch eingezeichnet wird.

«extend» - Beziehung: Erweiterungsstellen

- Mehrere Erweiterungsstellen (extension points) je Anwendungsfall möglich
- Namen von Erweiterungsstellen
  - müssen eindeutig sein
  - müssen nicht mit den Namen der erweiternden Anwendungsfälle übereinstimmen
- Beispiel: Kaugummiautomat



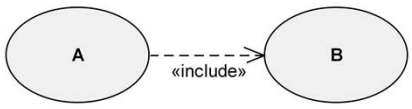
14

Wir können bei der extend-Beziehung noch **Extension Points** angeben, die definieren, an welcher Stelle denn unter Umständen etwas einfügt wird. Das wird in der Praxis, sag ich einmal, nicht oft verwendet. Zumindest im Diagramm sieht man es so gut wie nie. Der Sinn der Extension Points ist es, festzulegen, wo genau im Standardablauf die Erweiterung eingeführt wird.

Erinnern wir uns an das Kaugummi-Beispiel: Hier hatten wir vier Schritte im Standardablauf: „Münze einwerfen“, „Hebel drehen“, „Klappe öffnen“, „Kaugummi entnehmen“. Ich könnte nun zwischen „Hebel drehen“ und „Klappe öffnen“ einen Extension Point definieren. Diesen Extension Point nenne ich „Automaten schütteln“. Diesen „Automaten schütteln“ Extension Point werde ich nicht immer durchführen, sondern nur wenn eine bestimmte Bedingung eintritt und diese Bedingung könnte lauten „Wenn keine fallenden Geräusche von Kaugummis zu hören sind“. Ich gebe den Extension Point im Use Case selbst an, während ich die Bedingung als Notiz zur Beziehung hinzufüge.

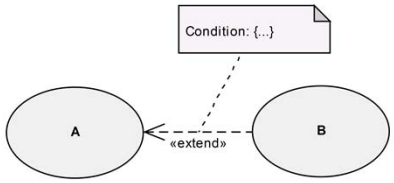
Analogien zu Programmiersprachen

- **<<include>>** entspricht Unterprogrammaufruf bzw. Makroexpansion




```
void B () { ... }
void A () {
  ...
  B();
  ...
}
```

- **<<extend>>** entspricht bedingtem Unterprogrammaufruf

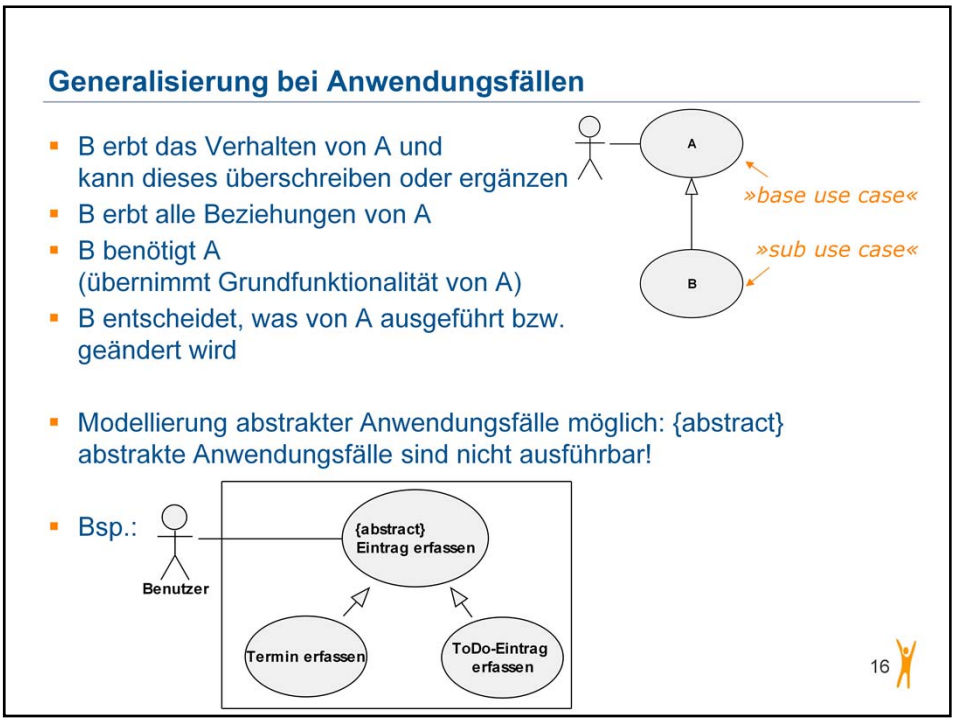



```
void B () { ... }
void A () {
  ...
  /* EP */
  if Condition then B();
  ...
}
```

- **Generalisierung** von Anwendungsfällen entspricht etwa dem **super**-Konstrukt von Java

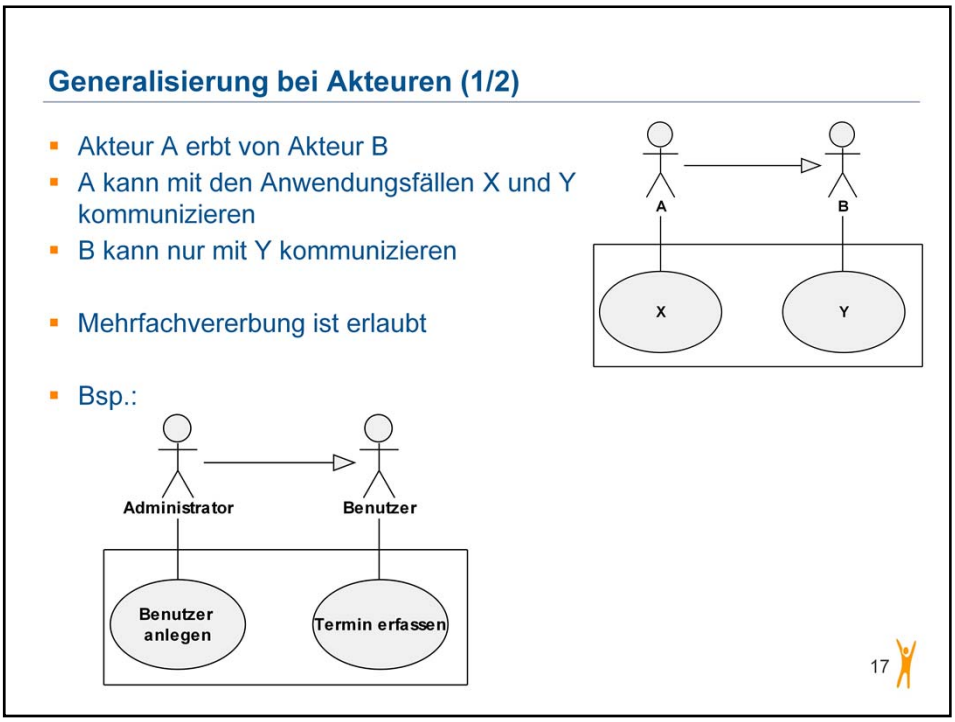
15

Hier sehen wir noch eine **Analogie zu Programmiersprachen**. Das ist jedoch nur für das Verständnis von include und extend gedacht. Sie werden niemals aus Anwendungsfällen einen Code ableiten. Dafür sind sie einfach nicht gemacht. Sie dienen zur Anforderungserhebung. Dies ist nur zum Verständnis, um den Unterschied von include und extends zu verdeutlichen. Bei include wird direkt ein Unterprogramm aufgerufen und bei extends wird ein Unterprogramm nur unter einer Bedingung aufgerufen.



Was wir dann auch noch haben, was Sie schon von den Klassendiagrammen kennen, ist die **Generalisierung**. Sie bedeutet: „B“ erbt das Verhalten von „A“ und kann dieses überschreiben oder ergänzen. Ganz wichtig jetzt in diesem Zusammenhang ist, dass „B“ auch alle Beziehungen von „A“ erbt. Das habe ich vorher bei include und bei extend nicht gesagt, wenn Sie aufgepasst haben. Nur bei der Generalisierung ist es wirklich laut Standard so, dass „B“ auch alle Beziehungen von „A“ erbt. „B“ übernimmt die Grundfunktionalität von „A“. „B“ entscheidet, was von „A“ ausgeführt wird bzw. was geändert wird.

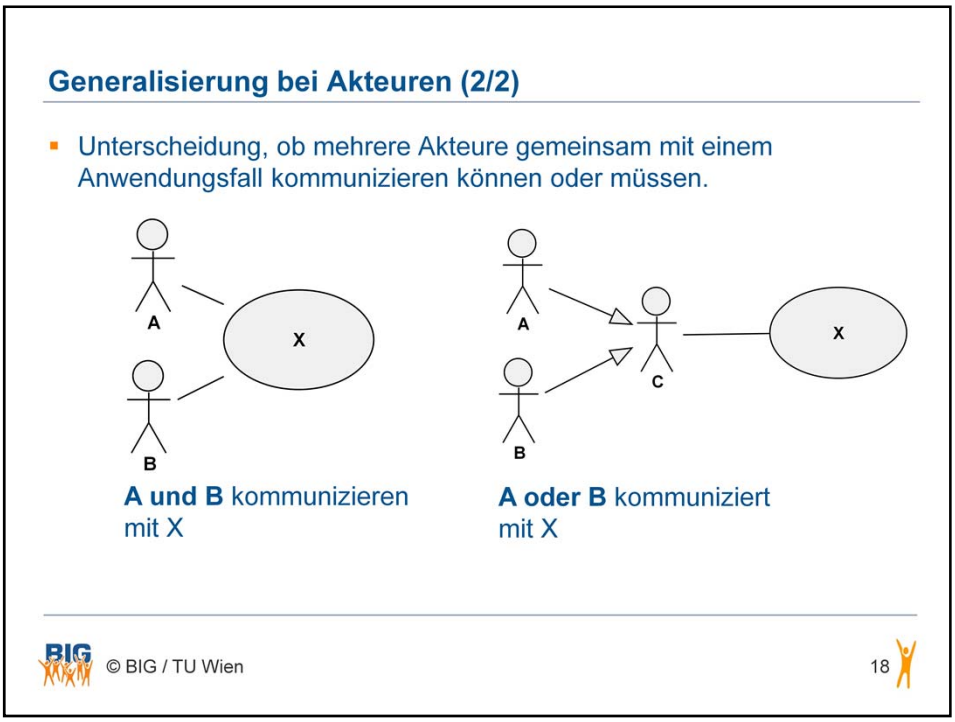
Wir haben hier auch ein Beispiel: Der Benutzer kann den Use Case „Eintrag erfassen“ ausführen. Dabei ist „Eintrag erfassen“ abstrakt, wobei das „abstract“ dasselbe bedeutet wie bei den Klassen. Es kann nicht sein, dass ich nur einen Eintrag erfasse, sondern ich muss dann unbedingt einen Termin erfassen oder einen ToDo-Eintrag erfassen in diesem Beispiel. Manche Schritte werden gleich sein, die führe ich oben bei „Eintrag erfassen“ an, und bei den Spezialisierung verfeinere ich das Verhalten. Der übergeordnete Use Case muss natürlich nicht abstrakt sein, ich kann ihn aber abstrakt definieren, so wie es hier vorgestellt ist.



Die **Generalisierung** gibt es auch **zwischen Akteuren**.

Hier haben wir ein Beispiel: „A“ erbt von „B“. „B“ ist mit „Y“ verbunden und „A“ ist mit „X“ verbunden. Wenn ich mir nur die Frage stelle: Wer kann den Use Case „X“ durchführen, dann lautet die Antwort „A“, und nur „A“. Wer kann „Y“ durchführen? – „Y“ kann von „B“ durchgeführt werden und nachdem „A“ ja eine Spezialform von „B“ ist, kann „A“ auch „Y“ durchführen.

Wenn wir uns das Beispiel unten anschauen, dann haben wir hier „Benutzer“, der mit „Termin erfassen“ verbunden ist und „Administrator“, der mit „Benutzer anlegen“ verbunden ist. „Benutzer anlegen“ kann nur vom Administrator durchgeführt werden. „Termin erfassen“ kann von einem Benutzer durchgeführt werden, aber nachdem ein Administrator auch ein Benutzer ist, kann natürlich auch ein Administrator „Termin erfassen“ durchführen.

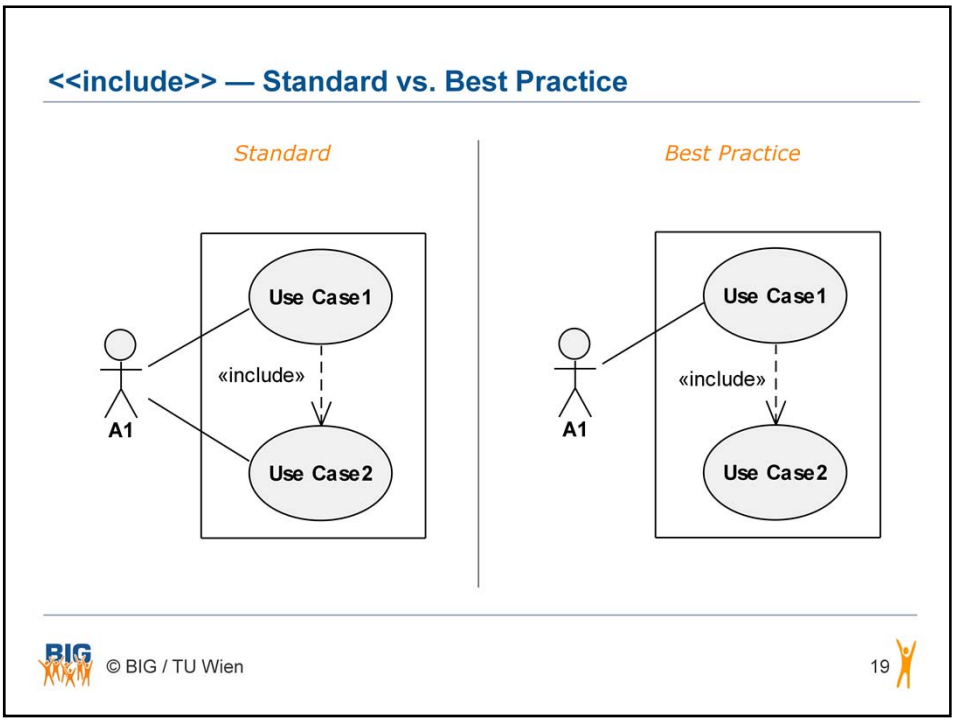


Sehr oft wird die Generalisierung bei Akteuren dazu verwendet, ein **entweder/oder** auszudrücken. D.h. ich möchte sagen, dass entweder die eine Rolle, oder die andere Rolle, also der eine Akteur oder der andere Akteur, einen Anwendungsfall durchführt.

Das wird in der Praxis sehr oft falsch gemacht, indem einfach der Use Case mit beiden Akteuren verbunden wird. Das ist das was wir links dargestellt haben. Hier sind „A“ und „B“ mit dem „X“ verbunden. Das bedeutet dann, dass ich beide Rollen dazu brauche um „X“ durchzuführen. „A“ und „B“ kommunizieren dann gemeinsam mit „X“.

Während ich auf der rechten Seite den Akteur „C“ einführe, den ich dann abstrakt machen würde, weil es keine direkte Instanz von „C“ geben kann. Das bedeutet dann, dass „A“ und „B“ von „C“ erben und ein „A“ eine Spezialform von „C“ ist und ein „B“ eine Spezialform von „C“ ist. Und einer alleine von beiden kann „X“ durchführen.

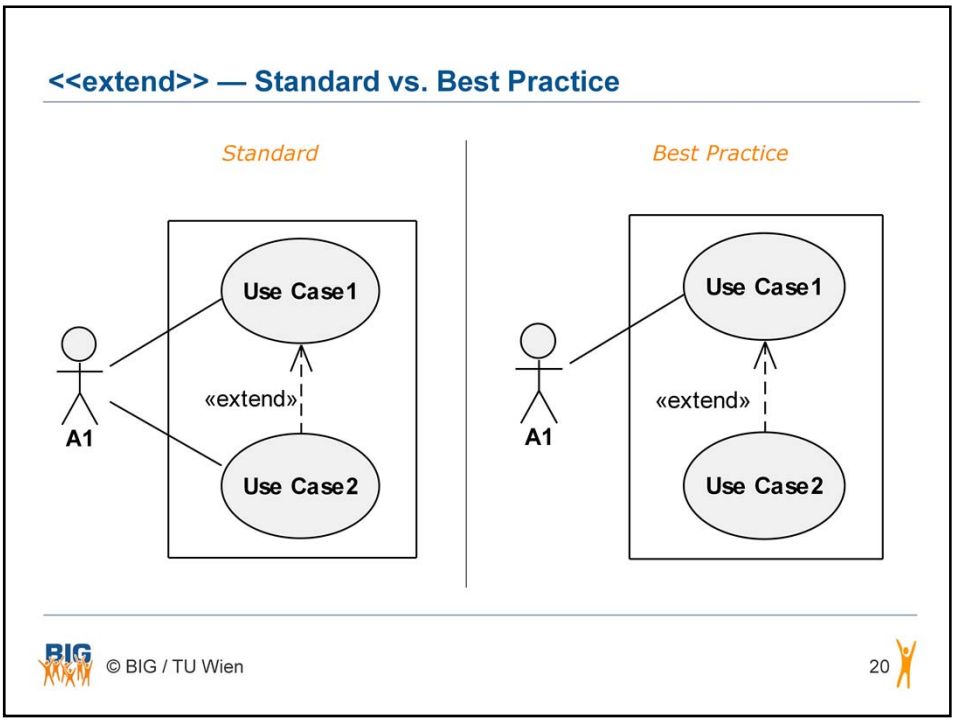
Die Modellierung von Anwendungsfällen bei denen zwei oder mehrere Akteure teilnehmen, ist Großteils nur für die Modellierung von Sozio-Ökonomischen-Systemen von Bedeutung. Weil bei IT-Systemen ist es meistens genau eine Rolle, die, die einen Anwendungsfall durchführen kann. Ausnahmen bei IT-Systemen bilden Fälle, wo Akteure gemeinsam gemäß eines Protokolls zusammenarbeiten sollen. Dies demonstriere ich später noch an einem nicht ganz ernst gemeinten Beispiel von einer Beichte mittels Skype.



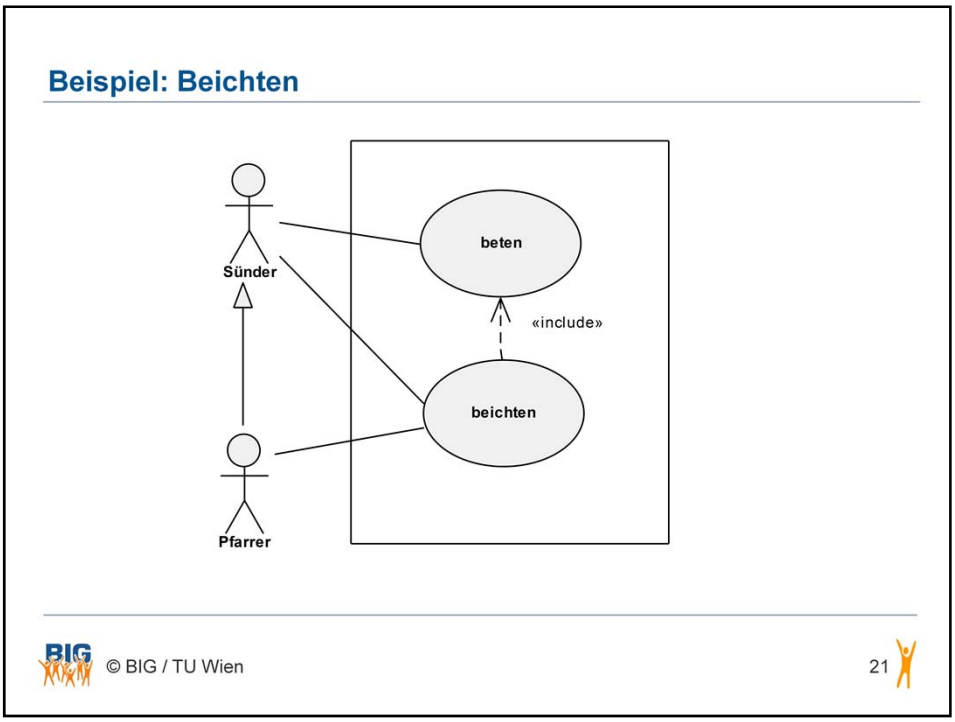
Wir kommen zur Verwendung von **include** im **Standard**-Fall bzw. im **Best Practice**-Fall.

Wir haben links die Standard-Notation dargestellt, für den Fall, dass der Anwendungsfall 1 den Anwendungsfall 2 inkludiert. Weiters sei angenommen, dass der Akteur „A1“ sowohl am Basis Use Case 1, als auch am inkludierten Use Case 2 teilnimmt. In diesem Fall muss eine Beziehung vom Akteur sowohl zum Basis Use Case 1, als auch zum inkludierten Use Case 2 aufgenommen werden. Denn laut Standard erfolgt eine Vererbung nur bei der Generalisierungs-Beziehung. Da wird die Beziehung vom Super Use Case zum Sub Use Case vererbt, nicht jedoch bei der include-Beziehung. Hier erbt der inkludierte Use Case nicht die Beziehung vom Basis Use Case zum Akteur. Dementsprechend ist die auf der linken Seite dargestellte Lösung laut Standard korrekt.

Im Best Practice Falle geht man jedoch meistens so wie auf der rechten Seite vor. Wenn ein Akteur mit einem Basis Use Case verbunden ist und der inkludierte Use Case keine Verbindung zu einem Akteur aufweist, dann nimmt man an, dass der inkludierte Use Case vom gleichen Akteur wie der Basis Use Case durchgeführt wird, bzw. wenn es mehr sind, von den gleichen Akteuren.



Die eben für include vorgestellte Unterscheidung zwischen Standard und Best Practice gilt ebenfalls für die **extend**-Beziehung. Auch hier müssen Sie laut Standard nicht nur zum Basis Use Case einen Akteur zuordnen, sondern auch zum erweiternden Use Case, selbst wenn der erweiternde Use Case vom selben Akteur durchgeführt wird. Im **Best Practice** Falle gilt jedoch, falls beim erweiternden Use Case kein Akteur zugeordne ist, so nimmt man an, dass der erweiternde Use Case vom selben Akteur durchgeführt wird, wie der Basis Use Case.



Ich habe schon vorher angedeutet, dass wir einen Anwendungsfall modellieren werden, bei dem es sich um eine Beichte zwischen zwei Akteuren handelt.

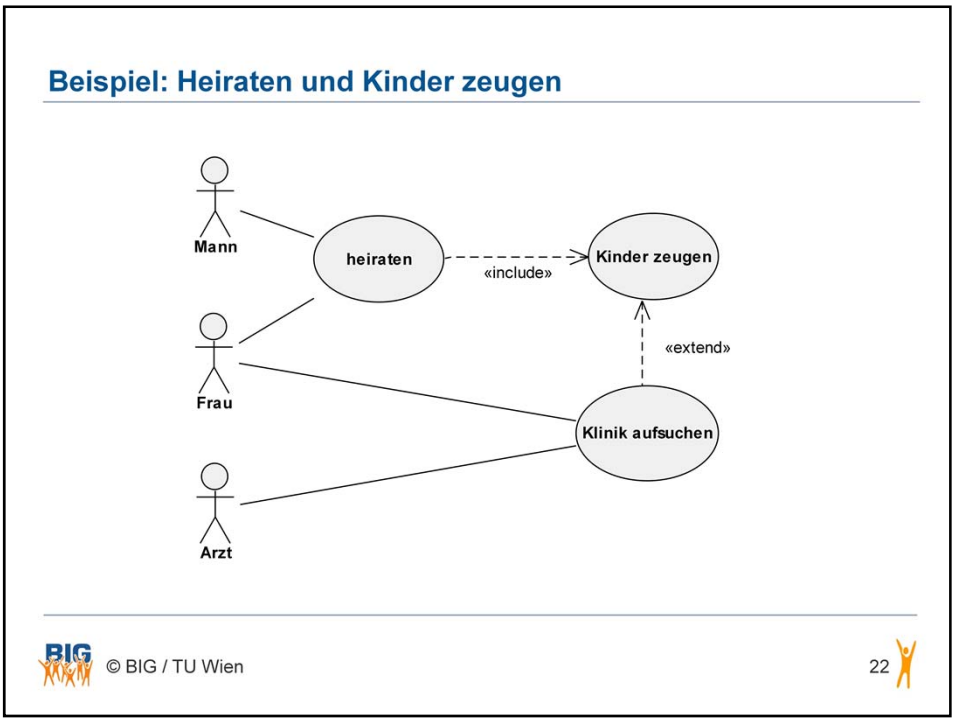
Unser Anwendungsfall heißt „beichten“. Daran beteiligt ist zunächst der Akteur „Pfarrer“ und der zweite Akteur, der immer involviert ist, ist unser „Sünder“. Beide wickeln über Skype die Beichte ab.

Die Kirche und die Informatik haben eines gemeinsam – sie laufen oft nach strengem Protokollen ab. Wenn Sie sich überlegen wie in der Kirche eine Messe abläuft, dann ist das ein streng vorgeschriebenes Protokoll. Jemand sagt etwas und dann hüpfen alle auf und im nächsten Moment knien sich wieder alle nieder, weil es eben nach strengen Regeln abläuft. Und genau so funktioniert aber in der Informatik auch vieles. Mehrere Akteure, die an einem Anwendungsfall beteiligt sind, kommen in der IT nur vor, wenn ich wirklich ein Protokoll zwischen ihnen vereinbaren muss.

Wir haben also jetzt unseren Anwendungsfall „beichten“. Ich weiß nicht, wie es Ihnen gegangen ist – ich war zwar schon länger nicht mehr beichten – aber bei mir war es immer zumindest so, damals, dass ich nie raus gekommen bin, ohne dass mir der Pfarrer am Schluss gesagt hat, was ich alles beten muss. Daher würde ich nicht von einem extend, sondern von einem include ausgehen. D.h. „beichten“ inkludiert „beten“.

Und wer betet denn jetzt? – Nur der Sünder. Weil, wenn mir der Pfarrer helfen würde – ich kann nämlich das „Gegrüßet seist Du Maria“ nicht mehr – dann könnte ich ja öfters beichten gehen. Aber nachdem ich das nicht kann, ist es jetzt so, dass mir der Pfarrer nicht hilft und wenn jedoch die Beziehung zwischen Akteuren und Anwendungsfällen bei der include-Beziehung automatisch vererbt werden würden, dann würde das ja bedeuten, dass der Pfarrer und der Sünder gemeinsam beten. Das entspricht aber nicht den Tatsachen. Denn es müssen beim Basis Use Case und beim inkludierten Use Case nicht immer die selben Akteure involviert sein. Und darum wird bei include laut Standard nicht vererbt. Deshalb muss ich zum Beten jenen Akteur zuordnen, der dies tatsächlich durchführt und dies ist der Sünder alleine.

Ich habe auch noch gehört, dass es unter den Pfarrern durchaus manche Sünder gibt. Was müsste ich jetzt machen, damit ich auch sicherstellen kann, dass jeder Pfarrer auch bei seinem Kollegen beichten kann? – Ein „Pfarrer“ ist auch ein „Sünder“. Ich könnte also – man wird es in der Praxis nicht machen – eine Generalisierungsbeziehung modellieren. Der „Pfarrer“ erbt von „Sünder“, er ist damit selbst auch ein „Sünder“ und so kann natürlich der eine Pfarrer beim anderen Pfarrer beichten.



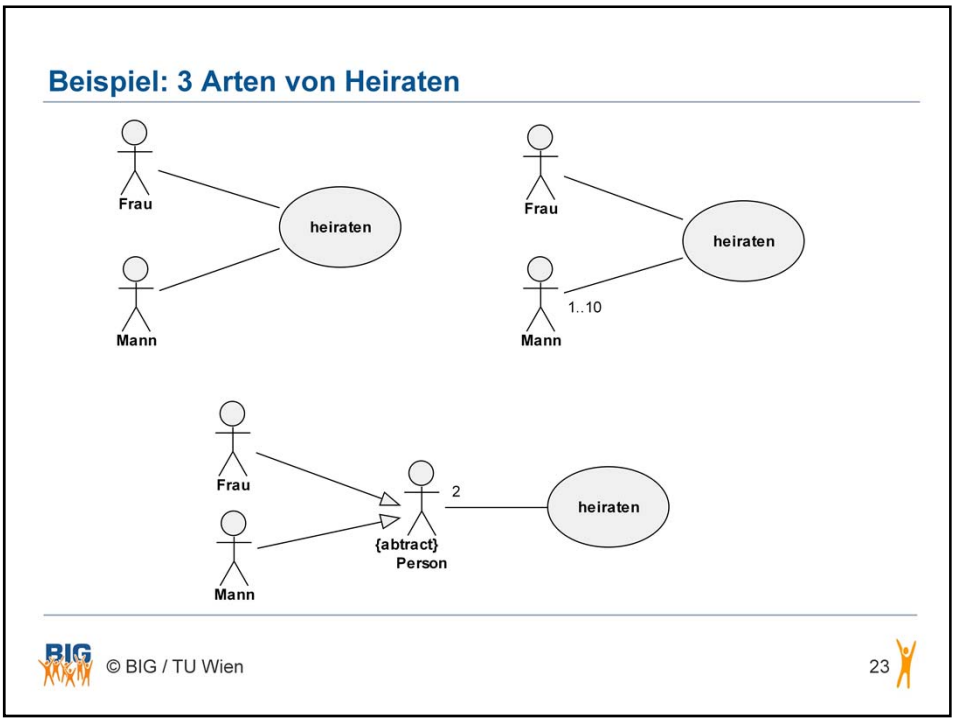
Einen zweiten Fall können wir auch noch durchmachen. Dieser stammt von einem Ihrer Kollegen, der bei dem Beispiel „Erklären Sie die include-Beziehung“ dieses Beispiel in der Übung gebracht hat.

Es gab die beiden Anwendungsfälle „heiraten“ und „Kinder zeugen“. Wir waren uns dann einig, dass dazwischen eine Beziehung steht. Es folgte eine längere Diskussion, ob es sich dabei um eine extends- oder ein include-Beziehung handelt und vor allem in welche Richtung die Beziehung geht. Wir haben uns dann an die Bibel gehalten und gesagt, dass „Heiraten“ das „Kinder zeugen“ inkludiert und nicht umgekehrt.

Die Akteure von „heiraten“ wurden mit „Mann“ und „Frau“ festgelegt. Dann würde ich nach Best Practice ja annehmen, dass diese beiden Beziehung vererbt werden und damit „Mann“ und „Frau“ auch mit „Kinder zeugen“ interagieren.

Jetzt sage ich aber: So einfach ist das nicht immer. Es könnt ja einen weiteren Anwendungsfall „Klinik aufsuchen“ geben, aber nur zu einem Zweck, nämlich um Kinder zu zeugen, also für eine künstliche Befruchtung. Das wird mit einer extends-Beziehung ausgedrückt. An diesem Use Case sind dann ein „Arzt“ und eine „Frau“ beteiligt. Wer macht jetzt aber das „Kinder zeugen“? Wenn nämlich alles immer nach „Kinder zeugen“ vererbt wird, dann funktioniert jedes „Kinder zeugen“ nur mehr unter der Aufsicht eines Arztes und das ist jetzt nicht gerade das, was wir wollen.

Nehmen Sie mir bitte dieses Extrembeispiele nicht übel. Ich wollte mit diesem Extrembeispiel nur zeigen, dass es bei include und bei extend nicht immer wünschenswert ist, dass automatisch vererbt wird. Und darum sieht der Standard bei include und bei extends keine Vererbung der Beziehungen vor.

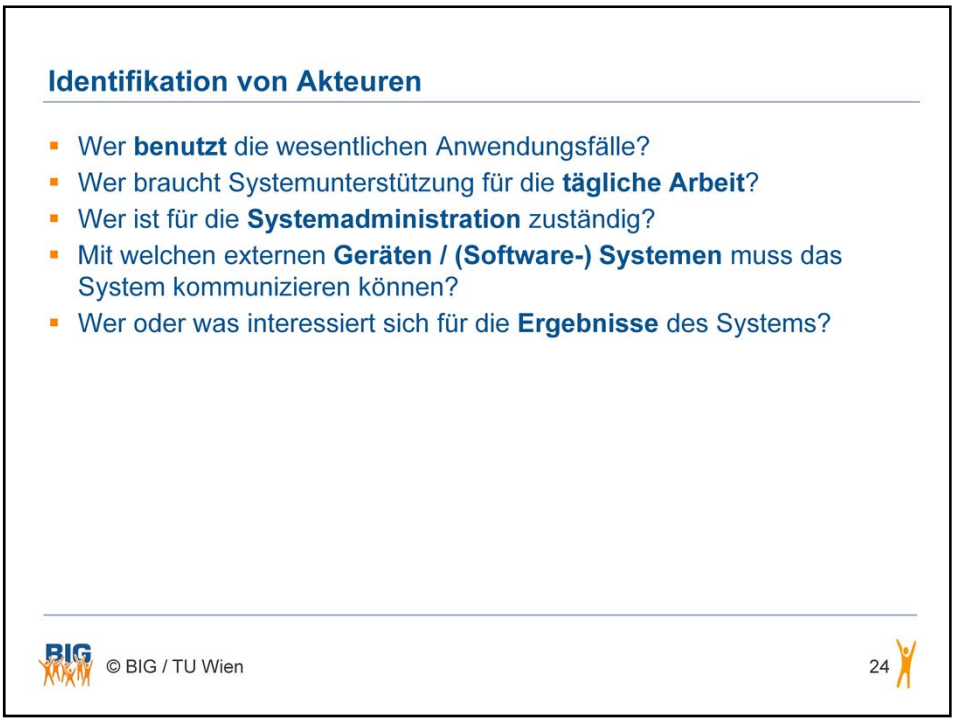


Wir wollen unser Beispiel Heiraten weiterführen.

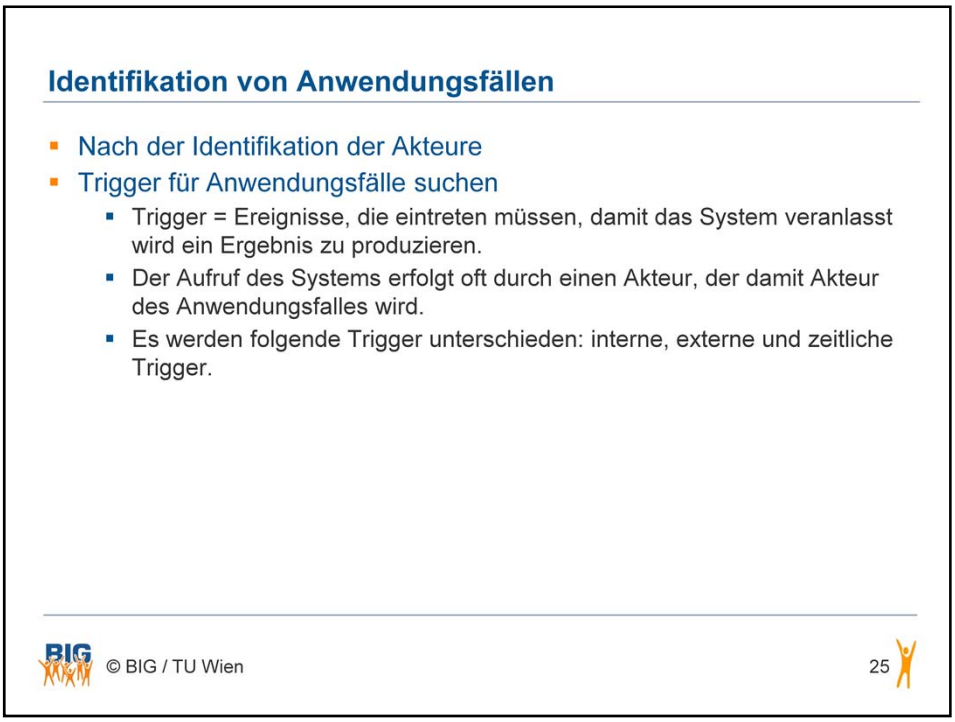
Also wir haben den Anwendungsfall „heiraten“ und daran beteiligt sind wiederum die beiden Akteure „Mann“ und „Frau“. Wie verändert sich dieses Bild, wenn wir die gleichgeschlechtliche Ehe erlauben?

Dazu verwenden wir die Generalisierung und Multiplizitäten. Dann führen wir den abstrakten Akteur „Person“ ein und von diesem erben die Akteure „Mann“ und „Frau“. Und wir haben gesagt, dass wir Multiplizitäten bei den Beziehungen zwischen Akteuren und Anwendungsfällen verwenden können. Damit können wir angeben, dass 2 Personen am Anwendungsfall „heiraten“ beteiligt sind. Das können jetzt ein Mann und eine Frau sein oder zwei Männer oder zwei Frauen.

Wir wechseln zum Abschluss noch einmal ins Lager der Amazoninnen. Eine Amazonin darf bis zu 10 Männer haben. Was müssen wir jetzt machen? Wir können dafür wieder Multiplizitäten verwenden. Am Anwendungsfall „heiraten“ ist genau eine „Frau“ beteiligt und 1 bis 10 „Männer“.

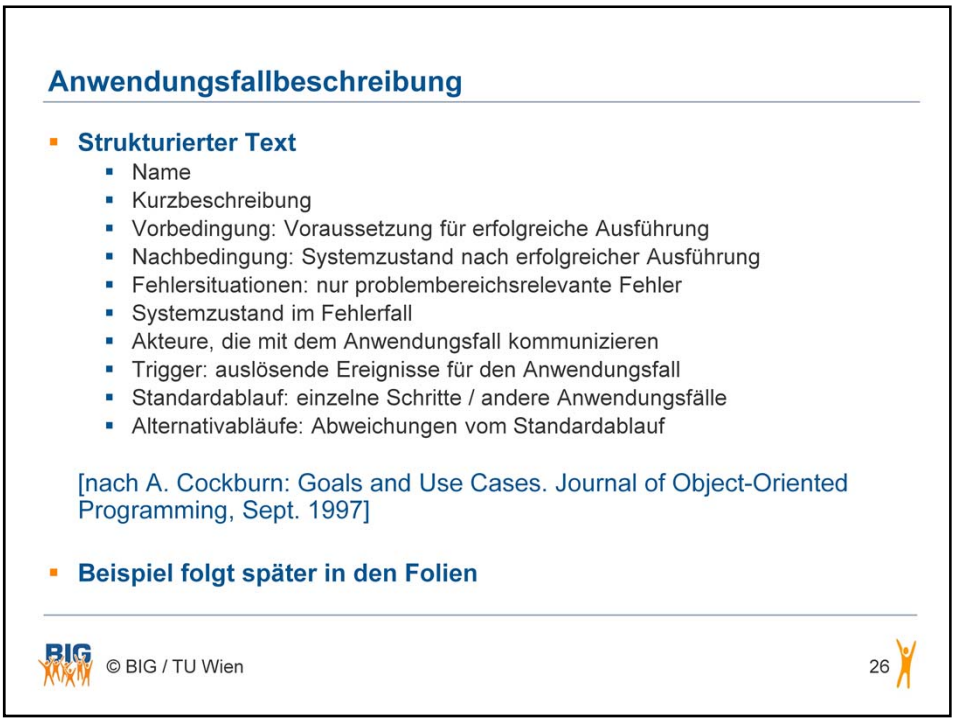


Wir kommen zur **Identifikation von Akteuren**. Das ist meistens gar nicht so schwer, denn man muss sich nur ein paar einfache Fragen stellen: Wer benutzt im Wesentlichen die Anwendungsfälle? Wer wird in einem System involviert sein? Wen brauche ich für die Systemadministration? Wer verwendet welche Teilsysteme oder Geräte? Wer oder was interessiert sich für die Systeme? Hierunter fällt auch das Management, das zwar das System nicht bedient, aber Reports erhält und sich damit auch für das System interessiert. Zusammenfassend kann man sagen, die Akteure findet man meistens relativ einfach.



Es ist eher die **Identifikation der Anwendungsfälle**, die schwierig ist in der Praxis. Hier muss man nach **Triggern** für Anwendungsfälle suchen. Was Sie als Kunde nie hören wollen ist „das kann mein System nicht“. Hier ist es wichtig, alle Trigger zu spezifizieren.

Man kann interne, externe und zeitliche Trigger unterscheiden. **Externe Trigger** sind jene, von denen ich meistens rede: Es passiert außerhalb des Systems etwas, das ich mit dem System abhandeln will. Es könnte aber auch innerhalb des Systems etwas passieren, das etwas anderes triggert, dann spricht man von einem **internen Trigger**. Und **zeitliche Trigger** triggern einen Anwendungsfall auf Grund von zeitlichen Angaben, z.B. soll am Monatsersten das Gehalt überwiesen werden.



Das Wichtigste bei den Anwendungsfällen ist die **Anwendungsfallbeschreibung**. Denn wenn Sie nur Ellipsen mit einem Namen aufzeichnen, so besteht der angegebene Name eines Anwendungsfalls ja maximal aus 3 Wörtern. In diesen 3 Wörtern werden Sie kaum ausdrücken können, was der Anwendungsfall genau bedeutet.

Daher benötigen Sie in eine Anwendungsfallbeschreibung. Diese ist nicht durch den Standard genormt. Üblicherweise enthält eine Anwendungsfallbeschreibung den **Namen**; eine ganz **kurze, abstrakte Beschreibung** in ca. zwei Sätzen; eventuell **Vorbedingungen** und **Nachbedingungen**, die gelten sollen.

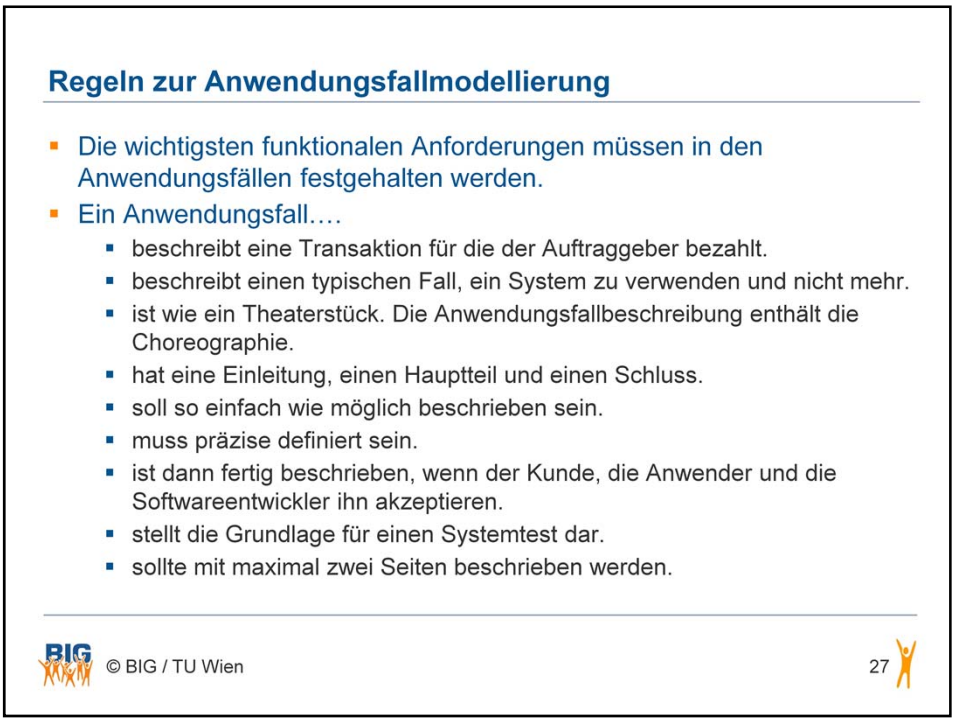
Vorbedingungen sind ganz, ganz wesentlich. Man setzt sie in der Praxis sehr oft ein. Sie könnten ja einen Anwendungsfall haben, der heißt „Login“. Der kommt sogar sehr oft vor und meistens ist es so, dass alle anderen Anwendungsfälle nur dann funktionieren, wenn Sie sich eingeloggt haben. Dementsprechend zeichnen viele dann immer z.B. ein, „zur Lehrveranstaltung anmelden“ includes „Login“. Dann inkludiert aber jeder Anwendungsfall „Login“. Wenn Sie das so machen, zeichnen Sie 100.000 Pfeilchen ein und das ist nicht Sinn der Sache. Daher verzichtet man oft auf diese include-Beziehung und nimmt in die Anwendungsfallbeschreibung als Vorbedingung auf, dass der Studierende eingeloggt ist. Das ist um einiges effektiver, was das Requirements Engineering betrifft. Bzw. beim „Login“ haben Sie die Nachbedingung, dass wenn alles gut geht, der Student angemeldet ist. D.h. Sie können Anwendungsfälle miteinander durch solche Vor- und Nachbedingungen verknüpfen.

In die Anwendungsfallbeschreibung werden auch **Fehlersituationen** aufgenommen, die auftreten könnten und die Ihnen schon bewusst sind. Probieren Sie nicht alles im Detail zu beschreiben. Weil wenn Sie da eine Beschreibung mit allen möglichen absurden Fällen, die auftreten können, machen, trägt das oft nicht zum Verständnis bei. Sie müssen sich immer darüber im Klaren sein, dass die Anwendungsfallbeschreibung zur Anforderungserhebung dient. Durch die Anwendungsfallbeschreibung soll nachher allen beteiligten Personen klar sein, welche Anforderungen es gibt und je mehr Sie an komplexen Dingen aufnehmen, desto unklarer wird es.

Desweiteren nimmt man die **Akteure** auf – die sieht man im Diagramm auch –, die **Trigger** und dann den **Standardablauf**. Der Standardablauf ist die Sequenz von Transaktionen, die der entsprechende Akteur durchführt. Erinnern Sie sich immer an die Beschreibung vom Kaugummiautomaten, die gehört beim Standardablauf hin: Münze rein, Hebel nach rechts, Klappe auf, Kaugummi raus.

Weiters werden a**lternative Abläufe** angeführt. Hier könnte man zusätzlich notwendige Schritte anführen oder eben auch alternative Abläufe, um ein und dieselbe Funktionalität zu erreichen. Sie könnten ja z.B. in einem Anwendungsfall die Lehrveranstaltungen vom Christian Huemer listen. Dazu können Sie vielleicht in einem System vorher den Christian Huemer auswählen, und dann sagen „Lehrveranstaltung listen“, dann bekommen Sie alle Lehrveranstaltungen von ihm angezeigt. Es könnte aber auch sein, dass Sie eine Suchmaske für Lehrveranstaltungen benützen und hier in einem Feld einen Lehrenden eintragen können und auch so zu den Lehrveranstaltungen von Christian Huemer kommen. Das sind zwei unterschiedliche Abläufe, die dem User geboten werden, um letztendlich zum selben Ergebnis zu kommen. Solche Dinge sollte ich hier in der Anwendungsfallbeschreibung erheben.

Ein Beispiel für eine Anwendungsfallbeschreibung, die diesem vorgeschlagenen Template folgt, finden Sie in den hinteren Folien zum CALENDARIUM. Hier finden Sie eine Anwendungsfallbeschreibung für den Anwendungsfall „Zeugnis abholen“. Nehmen Sie dieses Beispiel als Muster für Ihre eigenen Übungsbeispiele her.



Wir kommen zu den wichtigsten **Regeln** für die Anwendungsfallmodellierung.

Die **wichtigsten funktionalen Anforderungen** müssen in den Anwendungsfällen festgehalten werden.

Ein Anwendungsfall beschreibt eine **Transaktion**, für die der Auftraggeber bezahlt, nicht das, was der Informatiker implementiert, sondern das, was der Kunde bezahlt.

Ein Anwendungsfall beschreibt einen **typischen Fall ein System zu verwenden** und nicht mehr. Oft wollen Ihre Auftraggeber jeden Sonderfall genau beschreiben, obwohl Sie neu in der Domäne sind und noch nicht viel Ahnung haben. Deshalb probieren Sie die typischen Fälle herauszufiltern.

Ein Anwendungsfall ist wie ein **Theaterstück**. Die Anwendungsfallbeschreibung enthält die **Choreographie**. Denken Sie hier wieder an die Choreographie für den Kaugummiautomaten, die Choreografie zwischen dem Kind und dem Automaten, die beschreibt, was das Kind durchführen muss.

Ein Anwendungsfall hat eine **Einleitung**, einen **Hauptteil** und einen **Schluss**. Das bezieht sich auf den Aufbau der textuellen Beschreibung eines Anwendungsfalles.

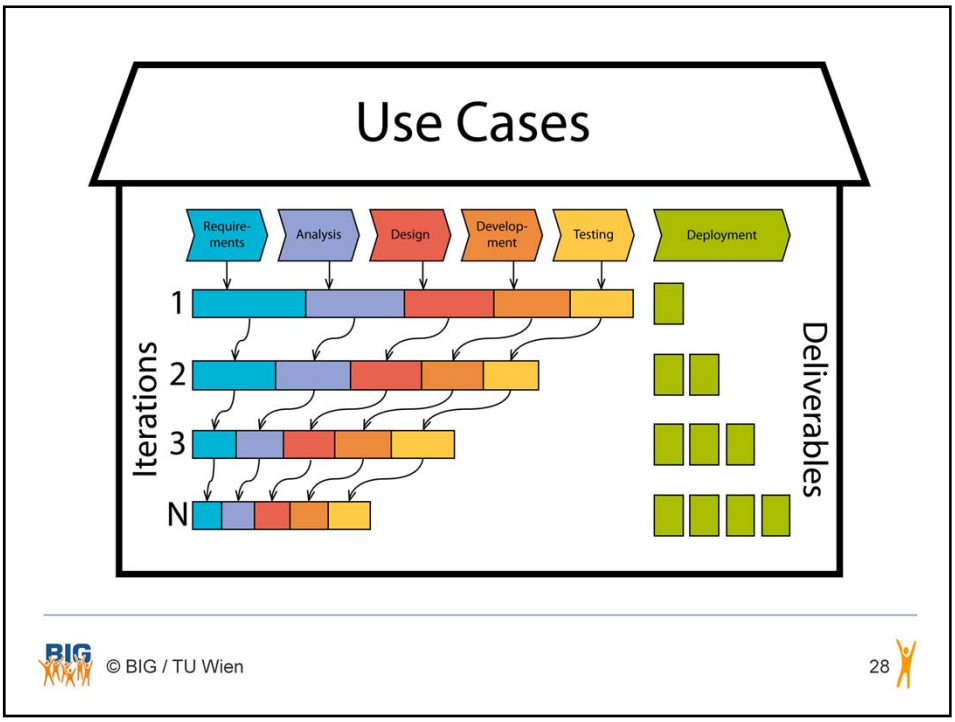
Ein Anwendungsfall soll **so einfach wie möglich** beschrieben sein.

Trotzdem soll der Anwendungsfall **präzise** definiert sein.

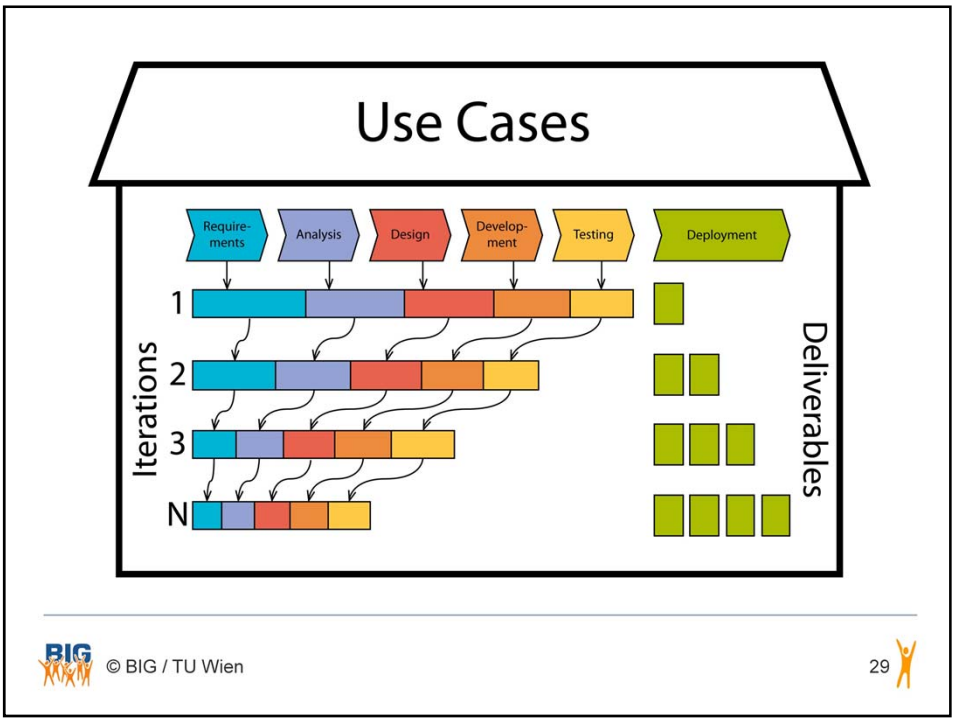
Ein Anwendungsfall ist dann fertig beschrieben, wenn der Kunde, die Anwender und die Softwareentwickler ihn **akzeptieren**. Das ist einer der wesentlichsten Sätze hier. Mit einem Anwendungsfall müssen alle einverstanden sein. Wenn den Anwendungsfall nur der Analyst mit dem Kunden definiert, dann wird er meistens von den Software-Entwicklern abgelehnt, weil sie ihn nicht verstehen. D.h. fertig ist ein Anwendungsfall dann, wenn er von den wichtigsten Stakeholdern akzeptiert wird. Das heißt nicht, dass die jetzt alle am Tisch sitzen müssen, aber die Beschreibung ist dann in Ordnung, wenn alle einverstanden sind und Ihre Zustimmung geben.

Und folgendes ist auch noch wichtig: Ein Anwendungsfall stellt die Grundlage für **Systemtests** dar. Weil wenn Sie die Anforderungen am Anfang mit ihren Anwendungsfällen beschreiben, dann werden Sie diese hoffentlich, wenn das System einmal implementiert ist, auch alle testen, um zu überprüfen, ob wirklich alles so funktioniert, wie es in den Anwendungsfällen beschrieben ist.

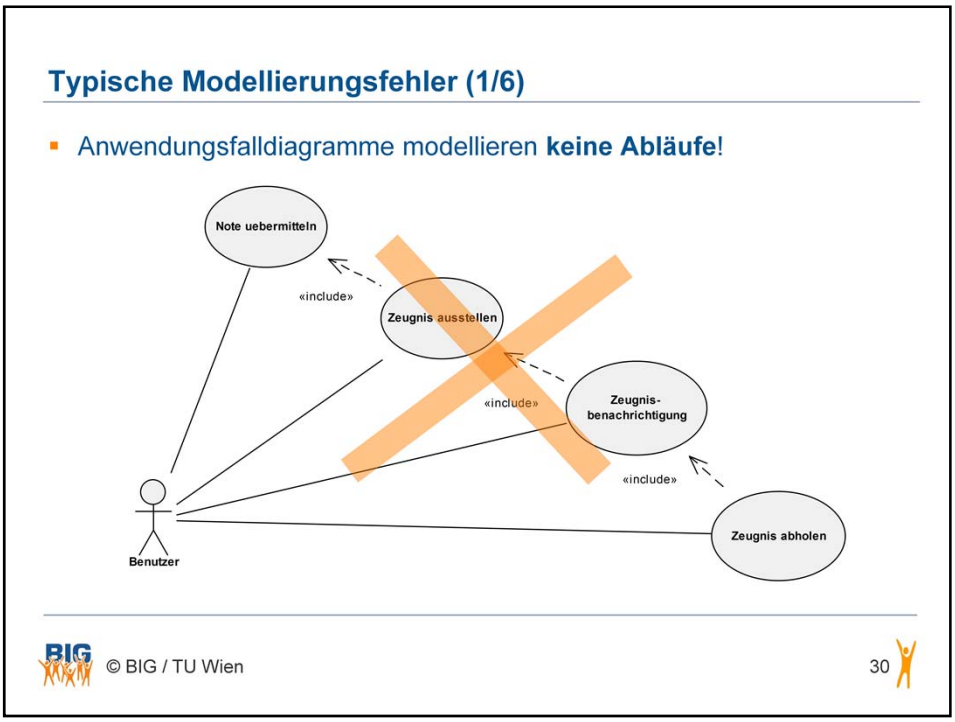
Ein Anwendungsfall sollte mit **maximal zwei Seiten** beschrieben werden. Nichts ist einfacher, als viele Seiten lang etwas zu beschreiben. Doch ein richtiges System hat verdammt viele Anwendungsfälle. Wenn Sie sich z.B. überlegen, wie viele Anwendungsfälle in TUWEL abgedeckt werden. Und wenn Sie für jeden Anwendungsfall dann 10 Seiten schreiben, dann haben Sie eine Requirements-Erhebung, die sich kein Entwickler durchliest. Daher formulieren Sie die Anwendungsfälle präzise und nicht wortgewaltig.



Hier haben wir etwas dargestellt, was ich schon mehrfach erwähnt habe. Der Anwendungsfall, so wie ich ihn jetzt beschrieben habe, entsteht während der **Anforderungserhebung**. Hier erheben Sie den Anwendungsfall und natürlich verwenden Sie diesen in den späteren Phasen des Softwareentwicklungsprozesses um zu überprüfen, ob Sie Ihre Anforderungen mit Ihrer Analyse, mit Ihrem Design, mit Ihrer Entwicklung und ganz wesentlich auch beim Testen entsprechend unterstützen.



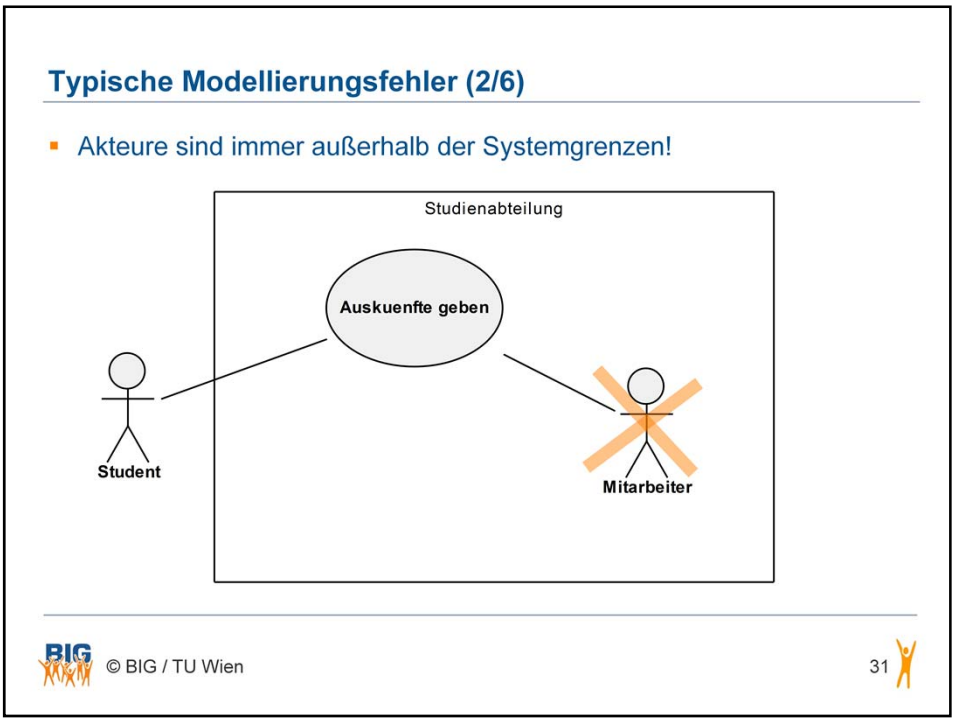
Es kann natürlich vorkommen, dass irgendetwas bei den Anwendungsfällen nicht stimmt. Oft erkennen Sie irgendwann im Developement, dass ein Anwendungsfall doch nicht eindeutig beschrieben wurde oder so nicht umgesetzt werden kann und der Software-Engineer diesen gar nicht unterschreiben hätte sollen. Der Software-Engineer muss dann noch eine Rückfrage stellen. Daher haben wir ein **iteratives Software Engineering**. D.h. der Use Case wird hier unter Umständen noch einmal adaptiert, weil sich Unklarheiten ergeben.



Abschließend kommen wir zur Vorstellung von typischen Modellierungsfehlern, wie Sie immer wieder bei der Anwendungsfallmodellierung vorkommen.

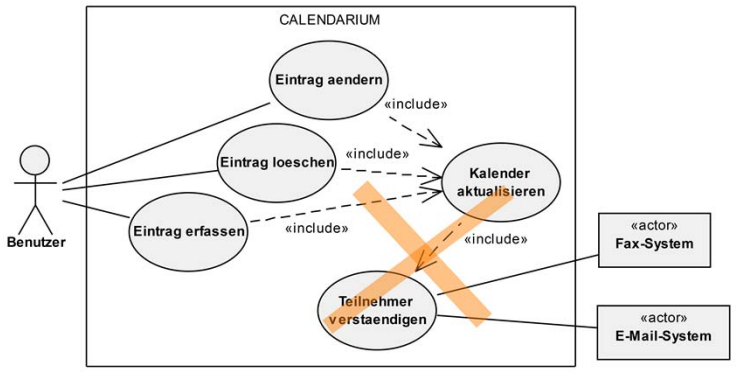
Auf dieser Folie sehen Sie einen typischen Fehler, bei dem mit Hilfe von Anwendungsfällen **Abläufe** modelliert werden. Das ist nicht im Sinne der Anwendungsfallmodellierung. Daher denken Sie nicht in Abläufen wie „Zeugnis abholen“ inkludiert „Zeugnisbenachrichtigen“, „Zeugnisbenachrichtigen“ inkludiert dass ein Zeugnis ausgestellt wird, „Zeugnis ausstellen“ inkludiert, dass eine Note übermittelt wird usw. Wenn Sie so denken, dann haben Sie einen Netzplan und nicht eine Anwendungsfallbeschreibung.

Es sei angemerkt, dass in der Praxis nicht immer versucht wird alle möglichen include- und extends-Beziehungen zwischen Anwendungsfällen zu veranschaulichen. Denn in der Praxis gibt es sehr sehr viele Anwendungsfälle und wenn ich alle möglichen Beziehungen erhebe, so verliere ich sehr schnell den Überblick und das ist nicht im Sinne der Anforderungserhebung.



Dieser Fehler ist meiner Meinung nach trivial: **Akteure stehen immer außerhalb der Systemgrenzen**.

Typische Modellierungsfehler (3/6)



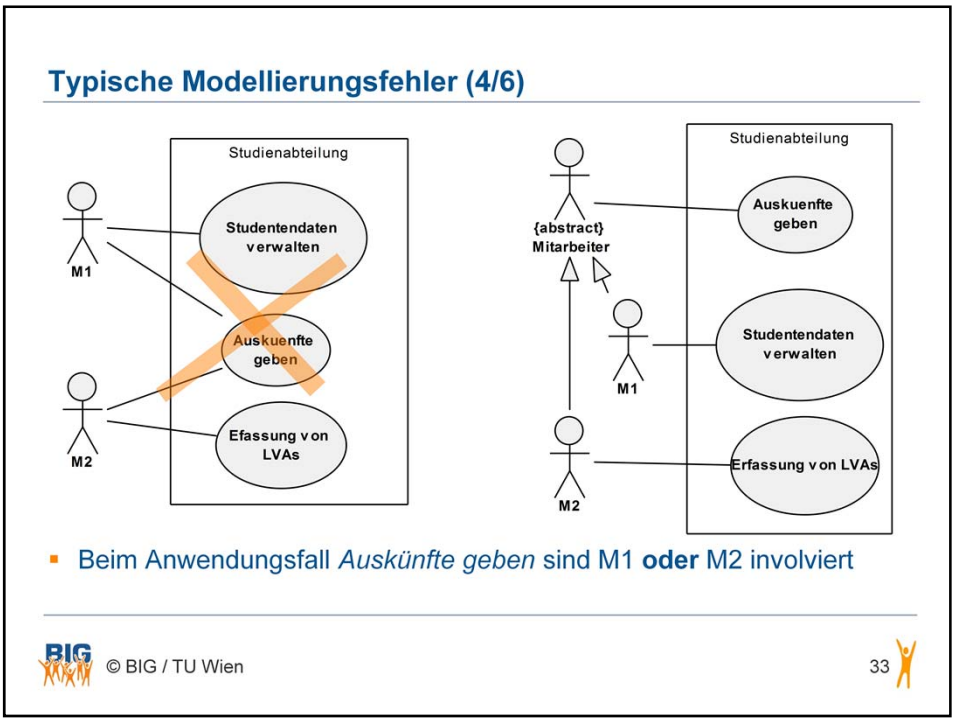
Obwohl der *Ablauf* so ist:

1) Eintrag erfassen
2) Kalender aktualisieren
3) Teilnehmer verständigen

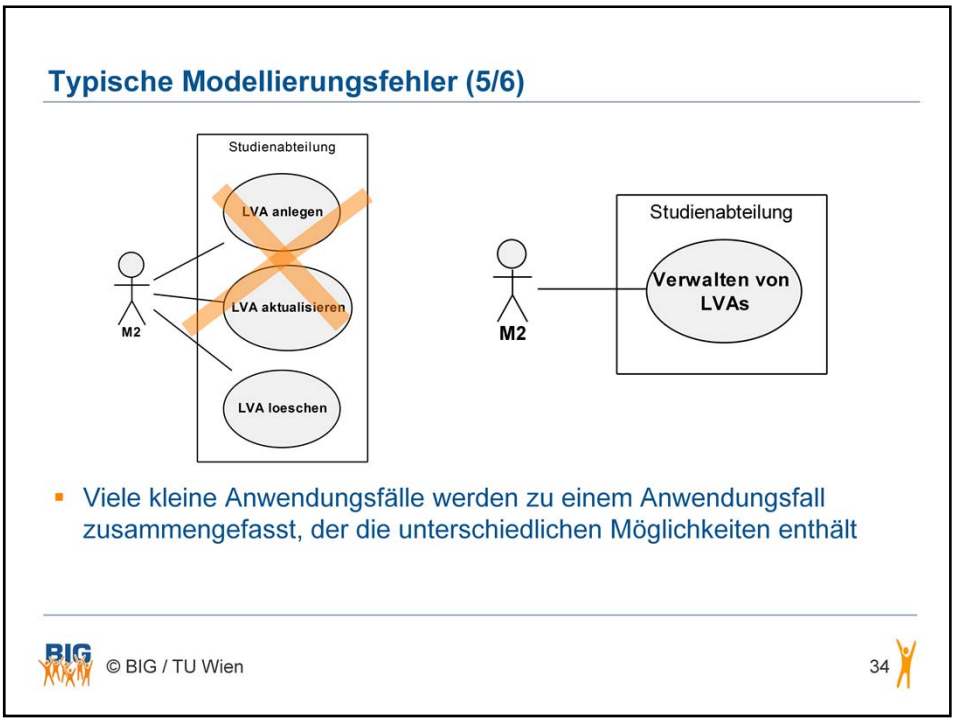
ist dieses Modell **falsch**, denn Teilnehmer verständigen ist nicht *Teil* von Kalender aktualisieren.

32

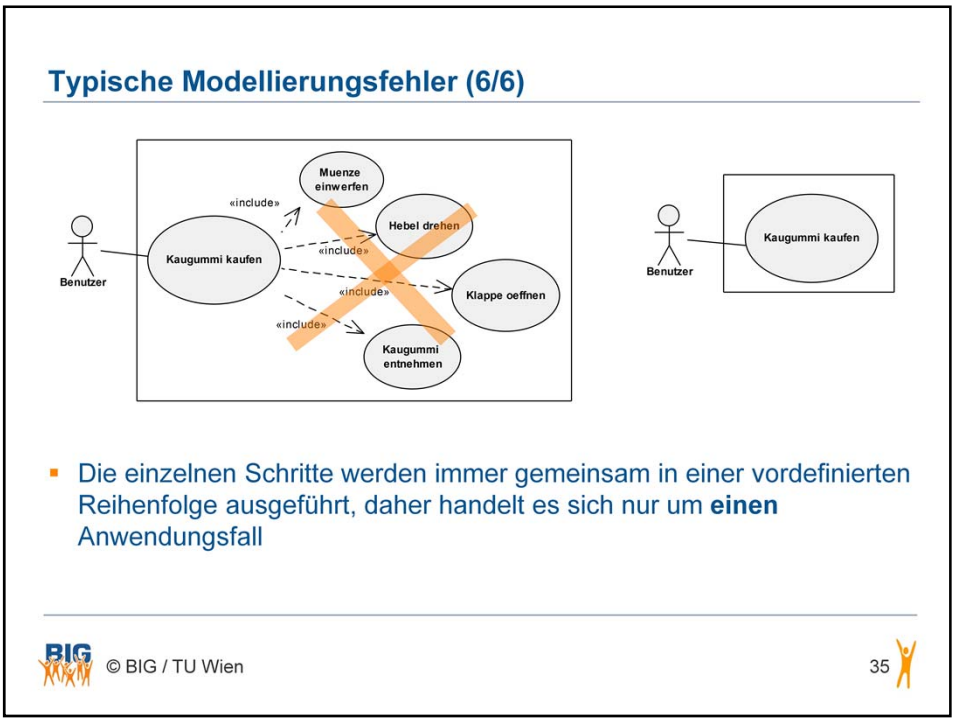
Das ist wieder so ein typischer Fehler, wo mit dem include ein **Ablauf** modelliert wird. Das Modell ist falsch, denn „Teilnehmer verständigen“ ist nicht Teil von „Kalender aktualisieren“. D.h. wenn ich den Kalender aktualisiere will, will ich nicht jedes Mal die Teilnehmer verständigen.



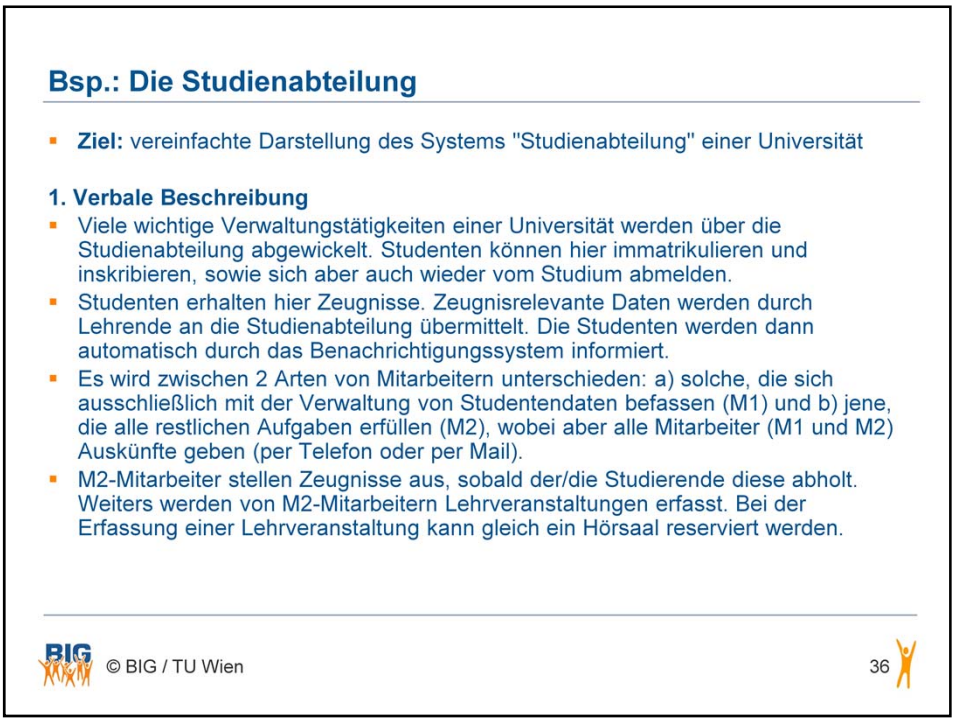
Hier möchten wir modellieren, dass den Anwendungsfall „Auskünfte geben“ entweder „M1“ **oder** „M2“ ausführt. Wenn dann allerdings beide **Akteure** „M1“ und „M2“ mit dem Anwendungsfall verbunden werden, bedeutet das, dass beide Akteure den Anwendungsfall gemeinsam ausführen, das wollte ich jedoch nicht ausdrücken. Daher ist, wie auf der rechten Seite richtig dargestellt, eine Generalisierung einzuführen, um ein „oder“ zwischen den beiden Mitarbeiter-Rollen „M1“ und „M2“ auszudrücken.



Was hier links dargestellt ist, ist nicht wirklich falsch. Wir haben hier die Anwendungsfälle „Lehrveranstaltung anlegen“, „Lehrveranstaltung aktualisieren“ und „Lehrveranstaltung löschen“. Das Vorgehen in den User-Interaktionen beim Löschen und beim Aktualisieren einer Lehrveranstaltung ist wohlweislich unterschiedlich. Daher sind das eigentlich drei verschiedene Anwendungsfälle. Andererseits, wenn Sie jeden dieser **Trivial-Anwendungsfälle** so intensiv beschreiben, dann erzeugen Sie wieder einen Stoß an Anforderungsdokumentation, den keiner liest. Und daher fasst man diese wirklichen Trivial-Anwendungsfälle, wie etwas Anlegen, Ändern, Löschen oft auch zu einem Anwendungsfall zusammen.



Dann kommen wir noch zu dem Punkt, dass **nicht alle Einzelschritte als Anwendungsfälle modelliert** werden sollen. Das ist so wie vorher beim Kaugummiautomat: „Kaugummi kaufen“ ist der Anwendungsfall und nicht die Einzelschritte „Münze einwerfen“, „Hebel drehen“, „Klappe öffnen“ und „Kaugummi entnehmen“. Wichtig ist, dass ein Anwendungsfall einen für den Nutzer messbaren Nutzen erzeugt und die Einzelschritte erzeugen üblicherweise keinen Nutzen, sondern nur die Summe davon, die ich mit dem Anwendungsfall beschreibe.



Damit sind wir am Ende der Theorie zu Anwendungsfällen angelangt. Auf den weiteren Folien finden Sie ein Musterbeispiel zur Anwendungsfallerhebung für das Beispiel einer Studienabteilung. Im Anschluss an dieses Beispiel finden Sie wieder eine Übersicht über die Notationselemente des Anwendungsfalldiagramms.

Damit sind wir am Ende unserer Vorlesung aus Objektorientierter Modellierung. Viel Erfolg noch bei den Übungen und bei den abschließenden Tests.

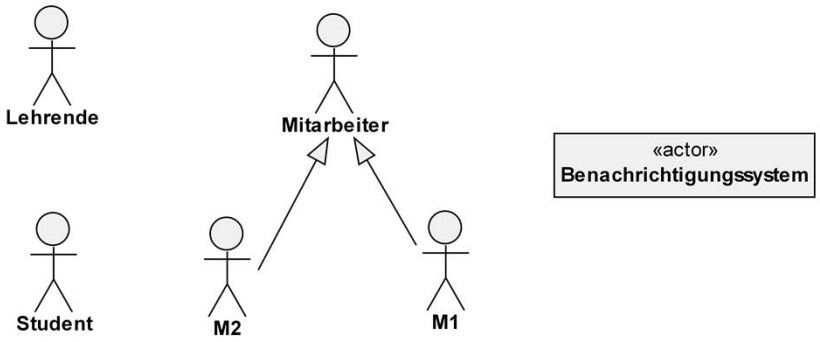
Bsp.: Akteure (1/2)

Wer sind die Benutzer?

- Viele wichtige Verwaltungstätigkeiten einer Universität werden über die Studienabteilung abgewickelt. **Studenten** können hier immatrikulieren und inskribieren, sowie sich aber auch wieder vom Studium abmelden.
- Studenten erhalten hier Zeugnisse. Zeugnisrelevante Daten werden durch **Lehrende** an die Studienabteilung übermittelt. Die Studenten werden dann automatisch durch das **Benachrichtigungssystem** informiert.
- Es wird zwischen 2 Arten von **Mitarbeitern** unterschieden: a) solche, die sich ausschließlich mit der Verwaltung von Studentendaten befassen (**M1**) und b) jene, die alle restlichen Aufgaben erfüllen (**M2**), wobei aber alle Mitarbeiter (M1 und M2) Auskünfte geben (per Telefon oder per Mail).
- M2-Mitarbeiter stellen Zeugnisse aus, sobald der/die Studierende diese abholt. Weiters werden von M2-Mitarbeitern Lehrveranstaltungen erfasst. Bei der Erfassung einer Lehrveranstaltung kann gleich ein Hörsaal reserviert werden.

Bsp.: Akteure (2/2)

- Wer sind die Benutzer?



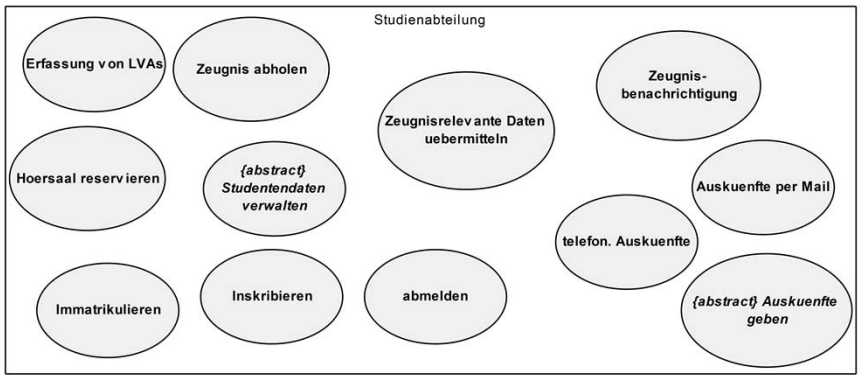
Bsp.: Anwendungsfälle (1/2)

Welche Funktionen bietet die Studienabteilung?

- Viele wichtige Verwaltungstätigkeiten einer Universität werden über die Studienabteilung abgewickelt. Studenten können hier **immatrikulieren** und **inskribieren**, sowie sich aber auch wieder vom Studium **abmelden**.
- Studenten **erhalten hier Zeugnisse**. Zeugnisrelevante Daten werden durch Lehrende an die **Studienabteilung übermittelt**. Die Studenten werden dann automatisch durch das Benachrichtigungssystem **informiert**.
- Es wird zwischen 2 Arten von Mitarbeitern unterschieden: a) solche, die sich ausschließlich mit der **Verwaltung von Studentendaten** befassen (M1) und b) jene, die alle restlichen Aufgaben erfüllen (M2), wobei aber alle Mitarbeiter (M1 und M2) **Auskünfte geben (per Telefon oder per Mail)**.
- M2-Mitarbeiter **stellen Zeugnisse aus**, sobald der/die Studierende diese abholt. Weiters werden von M2-Mitarbeitern **Lehrveranstaltungen erfasst**. Bei der Erfassung einer Lehrveranstaltung kann gleich ein **Hörsaal reserviert** werden.

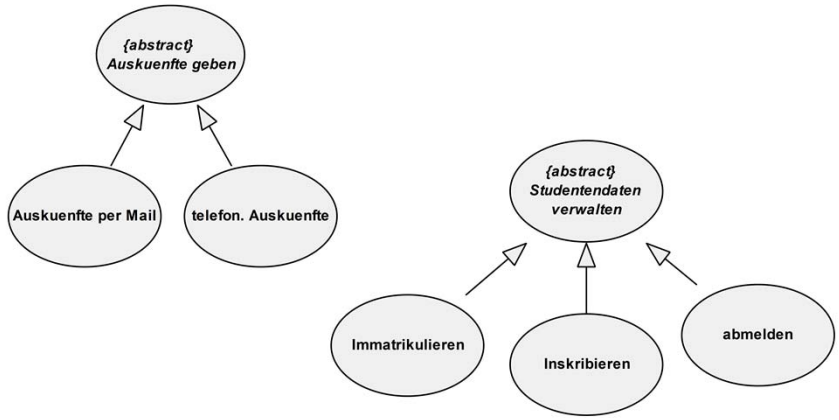
Bsp.: Anwendungsfälle (2/2)

- Welche Funktionen bietet die Studienabteilung?



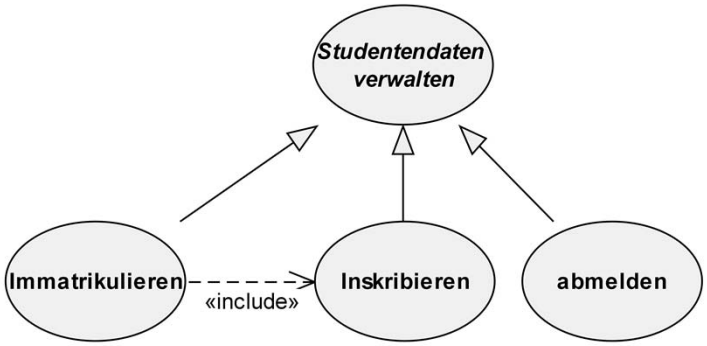
Bsp.: Generalisierungen

- Welche Anwendungsfälle können zusammengefasst werden?



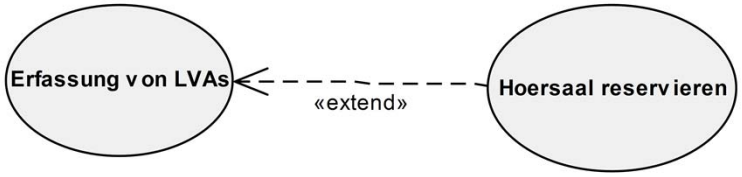
Bsp.: <<include>>

- Welche Anwendungsfälle sind Bestandteil eines anderen Anwendungsfalls?



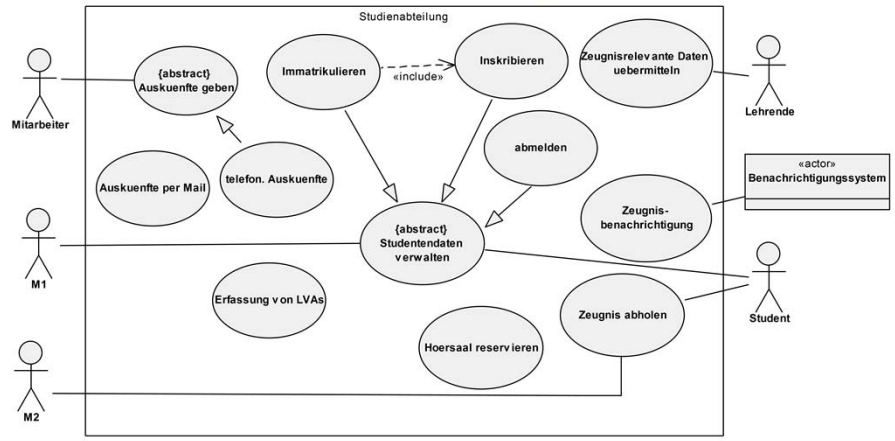
Bsp.: <<extend>>

- Welche Anwendungsfälle sind optional Bestandteile eines anderen?



Bsp.: Assoziationen

- Welche Akteure benutzen welche Anwendungsfälle?



Bsp.: Beschreibung (1/2)

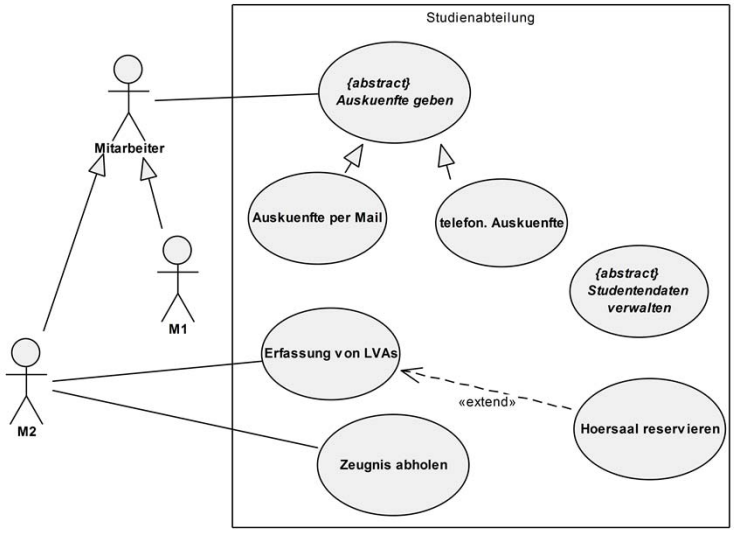
- **Name**: Zeugnis abholen
- **Kurzbeschreibung**: Wenn ein Student eine Prüfung abgelegt hat, kann er sich sein Zeugnis in der Studienabteilung abholen.
- **Vorbedingung**: Student hat LVA absolviert
- **Nachbedingung**: Zeugnis befindet sich in ausgedruckter Form beim Studenten
- **Fehlersituationen**: keine zeugnisrelevanten Daten übermittelt; das Zeugnis wurde bereits ausgestellt, der Studierende will kein kostenpflichtiges Duplikat
- **Systemzustand im Fehlerfall**: Student hat kein Zeugnis
- **Akteure, die mit dem Anwendungsfall kommunizieren**: Student, M2
- **Trigger**: Wunsch nach Zeugnis

Bsp.: Beschreibung (2/2)

- **Standardablauf:**
  - (1) Student geht in Studienabteilung.
  - (2) Student weist sich aus.
  - (3) Mitarbeiter überprüft, ob das Zeugnis vorhanden ist.
  - (4) Mitarbeiter überprüft, ob das Zeugnis bereits ausgedruckt wurde.
  - (5) Mitarbeiter druckt das Zeugnis aus.
  - (6) Student geht mit dem Zeugnis wieder.
- **Alternativabläufe:**
  - (5") Das Zeugnis wurde schon einmal ausgestellt – Mitarbeiter teilt Student mit, dass es gegen Kostenersatz noch einmal ausgestellt werden kann.
  - (6") Student geht in die Quästur und bezahlt für das Zeugnis.
  - (7") Student bekommt einen Beleg.
  - (8") Student geht mit dem Beleg in die Studienabteilung und ein Mitarbeiter druckt das Zeugnis aus.
  - (9") Student geht mit dem Zeugnis wieder.

Bsp.: Modellierungstipps (1/2)

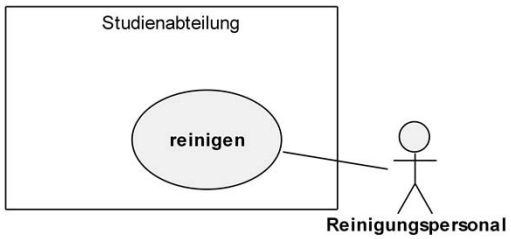
- Die Anwendungsfälle sollten auf der gleichen Abstraktionsebene sein.



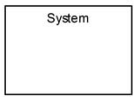
47

Bsp.: Modellierungstipps (2/2)

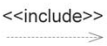
- Es sollen nur Anwendungsfälle und Akteure inkludiert werden, die relevant sind.
- Natürlich gibt es noch andere Personen, die etwas mit der Studienabteilung zu tun haben, die aber in diesem Kontext der Modellierung vernachlässigbar sind.
- Beispiel:



Anwendungsfalldiagramm - Elemente (1/2)

Name	Syntax	Beschreibung
Systemgrenze		Grenze zw. dem eigentlichen System und den Benutzern des Systems
Anwendungsfall		vom System erwartetes Verhalten
Akteur		Rolle der Systembenutzer

Anwendungsfalldiagramm - Elemente (2/2)

Name	Syntax	Beschreibung
Assoziation		Beziehung zwischen Anwendungsfällen und Akteuren
Generalisierung		Vererbungsbeziehung von Anwendungsfällen und Akteuren
extend		*A extends B*: opt. Verwenden von Anwendungsfall A durch Anwendungsfall B
include		*A includes B*: notw. Verwenden von Anwendungsfall B durch Anwendungsfall A

Zusammenfassung

- Sie haben diese Lektion verstanden, wenn Sie wissen …
- dass mit dem Anwendungsfalldiagramm das Verhalten eines Systems aus der Sicht der Benutzer beschrieben wird.
- dass an einem Anwendungsfalldiagramm Akteure und Anwendungsfälle beteiligt sind.
- dass bei einem Anwendungsfalldiagramm die Grenzen des Systems klar definiert sein müssen und sich die Akteure immer außerhalb des Systems befinden.
- wie das Zusammenspiel zwischen Akteuren und Anwendungsfällen aussieht.
- dass es sowohl für Anwendungsfälle als auch für Akteure Generalisierungsbeziehungen gibt.
- worin der Unterschied zwischen `<<include>>` und `<<extend>>` besteht.