


Softwaretechnik

Zusammenfassung zur Systematischen Entwicklung



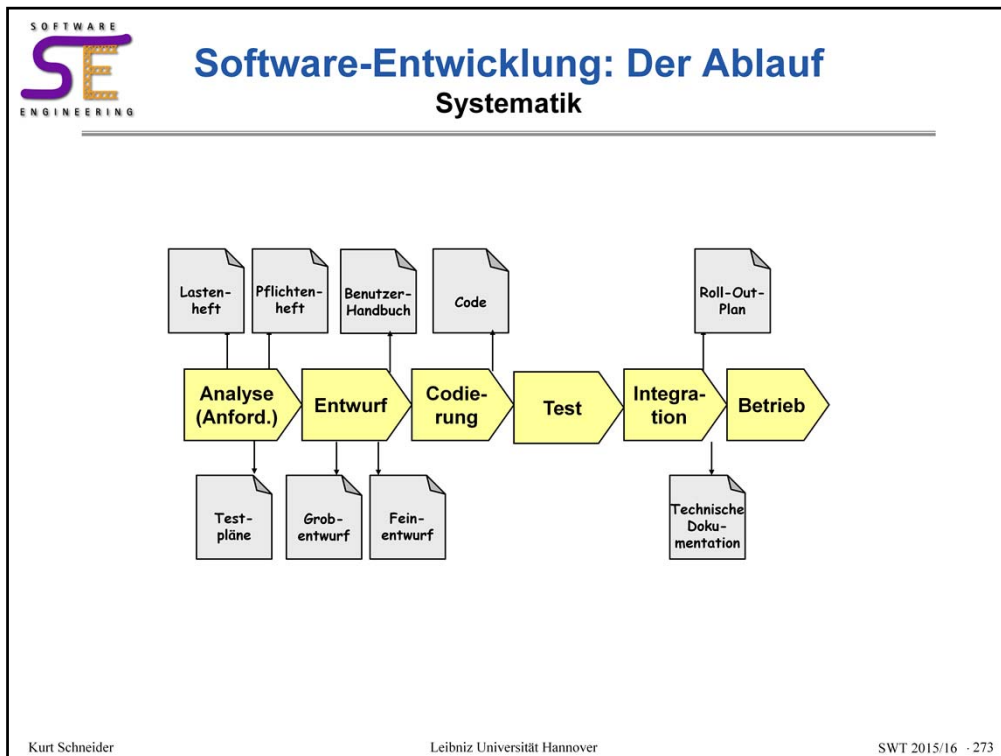
1. Wieso Software Engineering?
2. Vom Einzelkämpfer zum Großprojekt
- 3. Anforderungen und Test**
- 4. Entwurf: Strukturen und NF Eigenschaften**
- 5. Entwürfe notieren mit UML**
- 6. Design Patterns**
7. Management: *Technik und Projektmanagement*

Leibniz Universität Hannover

SWT 2015/16 - 272

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

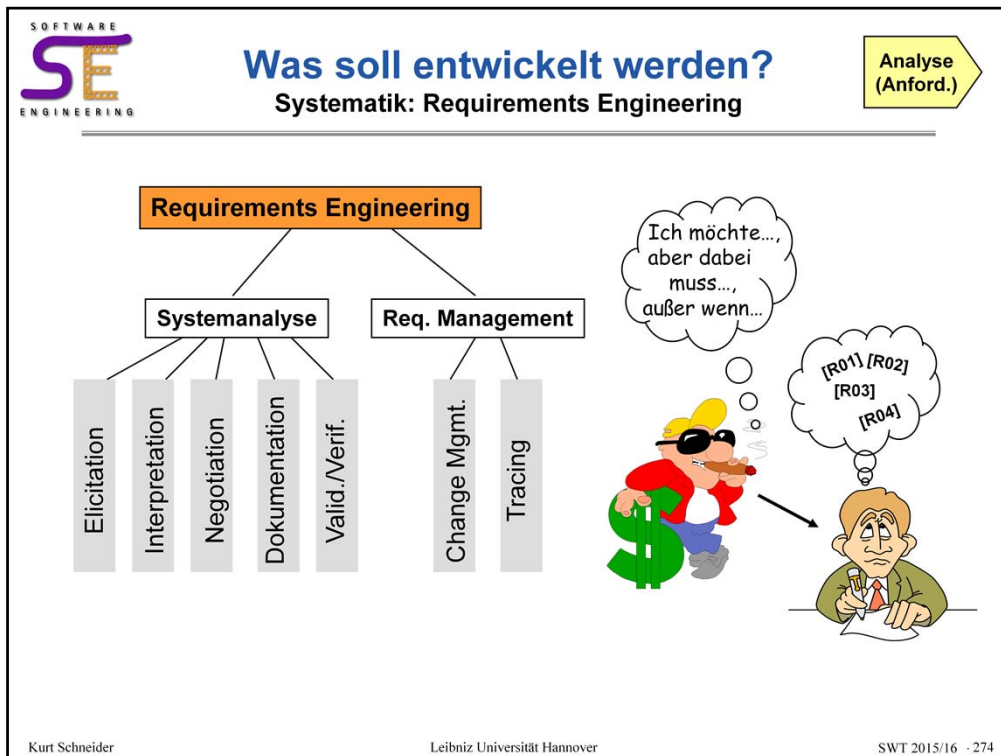
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Immer wieder wurde in der Vorlesung dieses einfache Ablaufmodell gezeigt. Eine Reihe von Aktivitäten (Mitte) bauen aufeinander auf. Dabei werden Dokumente erzeugt, die ebenso zur „Software“ gehören wie der eigentliche Programmcode.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

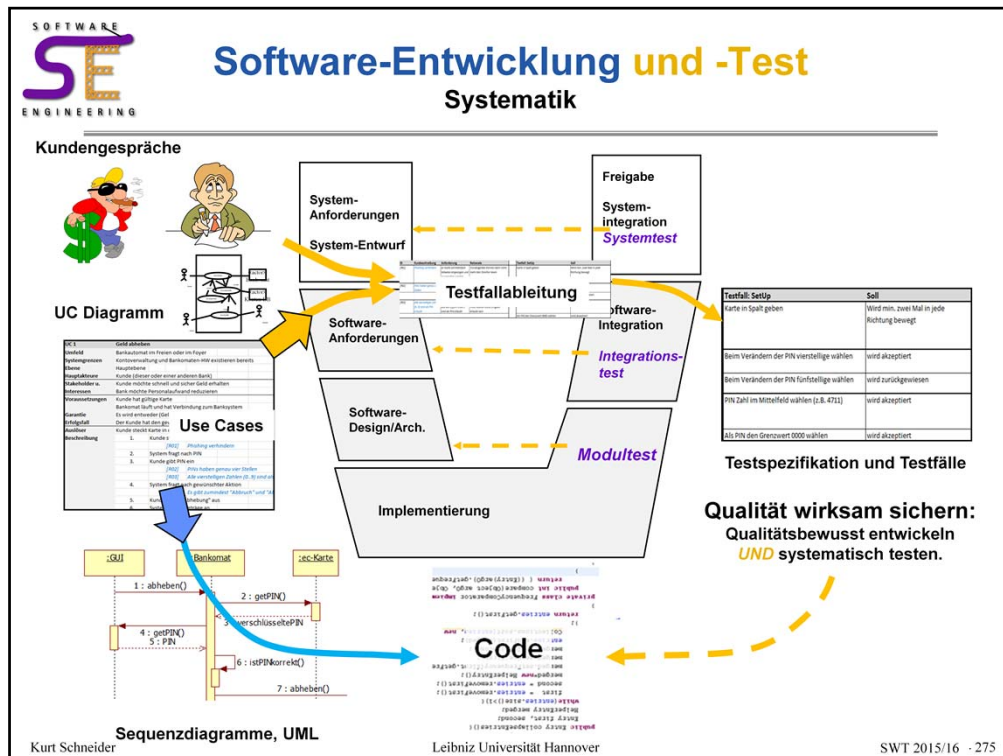


So beginnt ein Projekt: Zunächst müssen die Anforderungen erhoben werden. Dazu gibt es die Aktivitäten des Requirements Engineering. Im wesentlichen geht es hier darum, Anforderungen herauszufinden und von Missverständnissen zu reinigen, sowie sie im Projektverlauf zu verwalten.

Dabei spielen die Stakeholder und die Requirements Engineers eine große Rolle. Sie müssen sicherstellen, dass die wichtigen Anforderungen unverfälscht zu den Entwicklern gelangen. In manchen Unternehmen gibt es auch die Bezeichnung „Business Analysts“; der Begriff kann sehr ähnlich wie „Requirements Engineer“ verwendet werden, betont aber die wirtschaftlichen Aspekte (Unternehmensziele, Kosten und Erlöse) stärker als die eher technisch ausgerichtete Rolle des Requirements Engineers.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Diese Folie fasst die zwei zusammengehörigen Bewegungen zusammen:

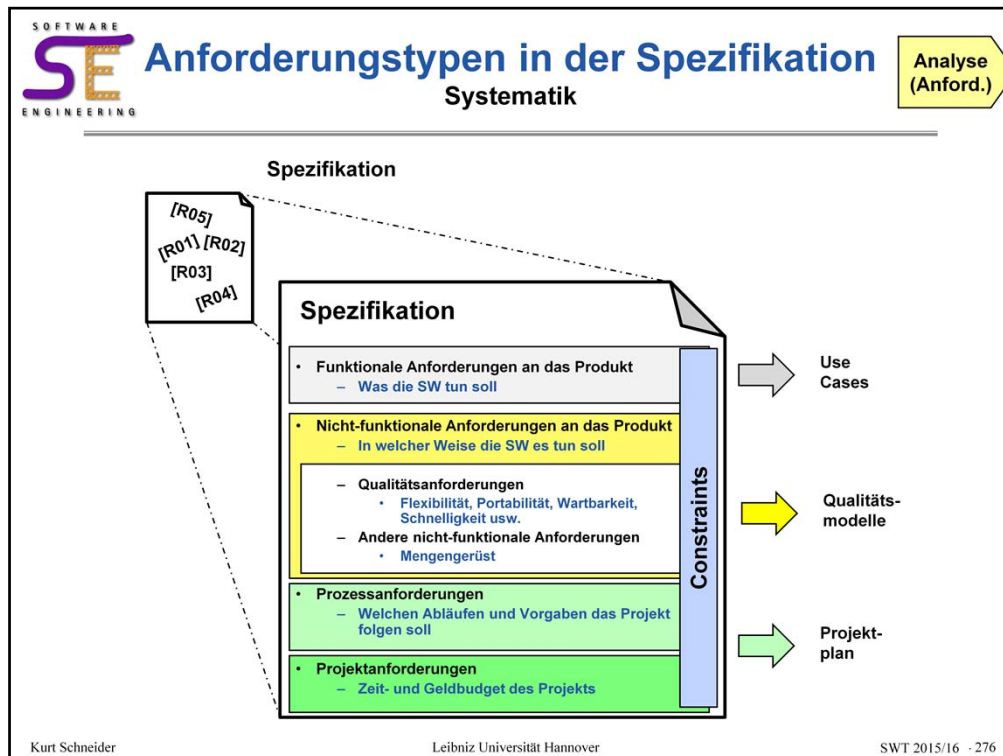
Im Zentrum stehen die Aktivitäten der Softwareentwicklung, hier V-förmig gezeichnet. Das zeigt die zeitliche Achse (x) und die Nähe zum Code (nach unten). Dabei geht man so vor:

- Mit Kundengesprächen werden die Anforderungen ermittelt.
- Dann verschafft man sich einen Überblick mit Hilfe eines Use Case-Diagramms
- Die eigentlichen Use Cases werden in Tabellenform notiert. Von hier aus arbeitet man in zwei Richtungen weiter.
- Konstruktiv (blau) werden UML-Modelle und speziell Sequenzdiagramme aus Use Cases abgeleitet. Die UML-Modelle können teilweise automatisch, sonst aber manuell in Code überführt werden.
- Qualitätssichernd kommen die Testfälle von der anderen Seite auf den Code zu (orange): Man leitet Testspezifikationen und letztlich konkrete Testfälle ebenso aus den Use Cases und den darin eingebetteten Anforderungen ab.

Testen und Qualitätssicherung gehören unbedingt zur professionellen Entwicklung. Fehler sind unvermeidlich; das ist nicht schlimm, wenn man sie früh findet und beseitigen kann. Liefert man aber fehlerhafte Software aus, sind der finanzielle und der Imageschaden dagegen sehr hoch.

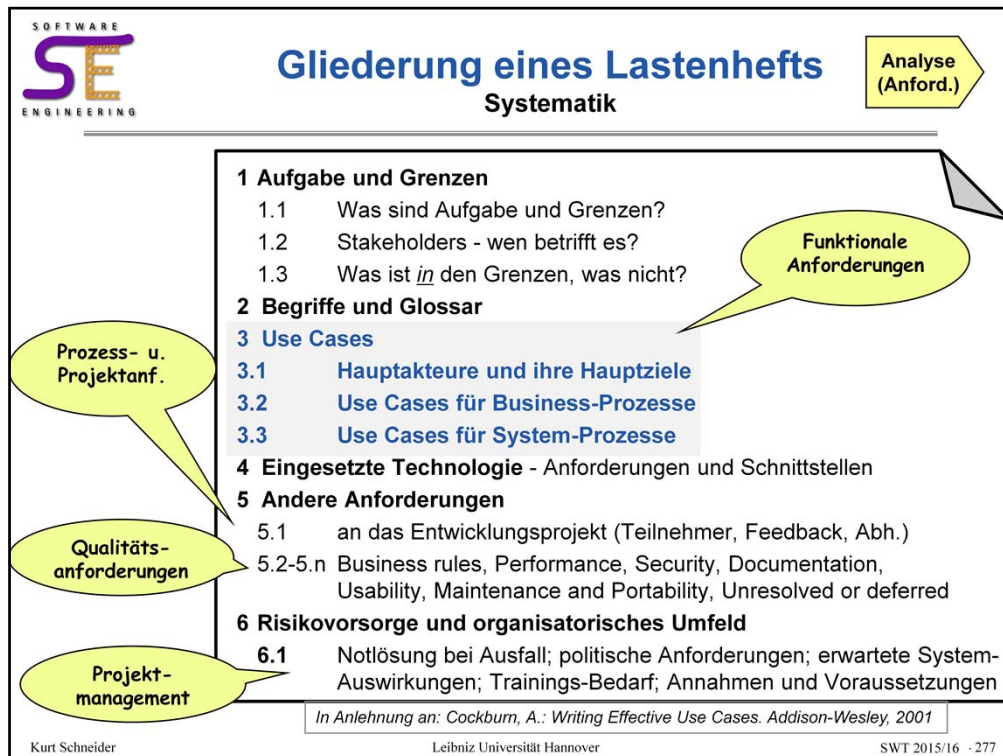
Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Nun noch einmal durch die wichtigsten Schritt:

- In der Spezifikation sollen alle Anforderungen stehen.
- Wir hatten gesehen, dass es davon verschiedene Typen gibt. Sie werden oft unterschiedlich behandelt und weiterverfolgt. Aber sie stehen alle in der Spezifikation.



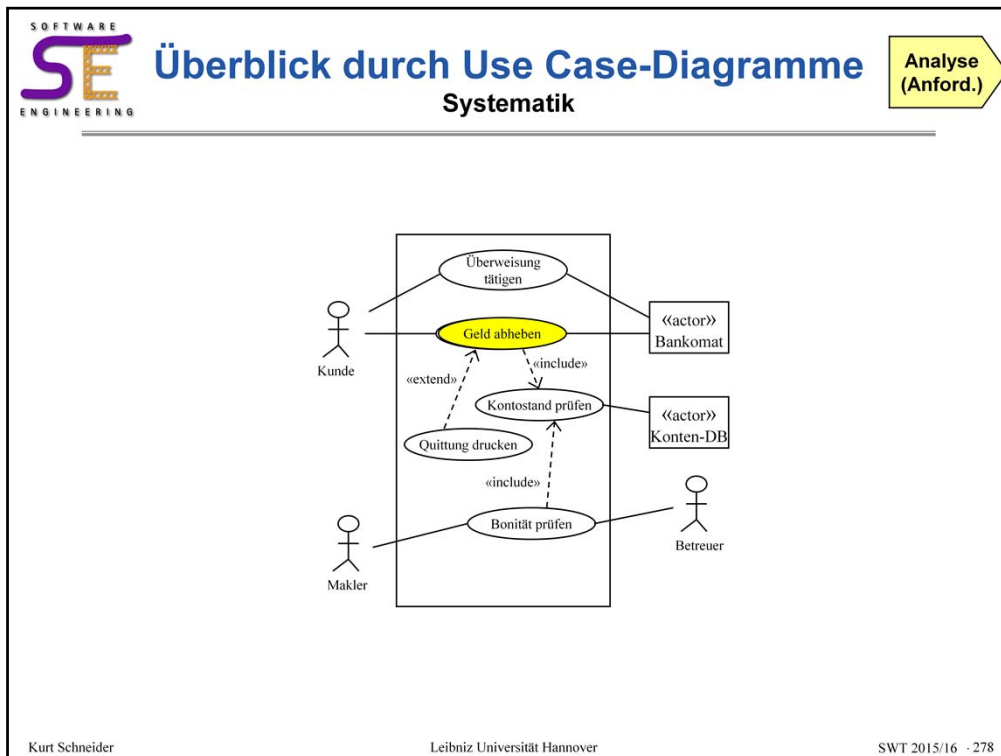
Dies ist eine mögliche Gliederung für ein Lastenheft-Dokument.

Man erkennt darin die verschiedenen Anforderungstypen.

- Über das Software-Projektmanagement wird in dieser Vorlesung noch gesprochen.
- Software-Qualität und die damit zusammenhängenden Anforderungen sowie Qualitätsmodelle werden im Sommersemester in der Vorlesung „Software-Qualität“ thematisiert.
- Funktionale Anforderungen werden mit Use Cases beschrieben. Wie es damit weitergeht, zeigen die folgenden Folien.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016


Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Einen Überblick über alle identifizierten Use Cases gibt ein Use Case-Diagramm. Es wird parallel mit den detaillierten Use Case-Tabellen entwickelt, zum Beispiel hier zum „Geld abheben“.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

SOFTWARE

ENGINEERING

Details im Use Case Template

UC Geld abheben (Teil 1)

Analyse
(Anford.)

UC 1	Geld abheben				
Umfeld	Bankautomat im Freien oder im Foyer				
Systemgrenzen	Kontoverwaltung und Bankomaten-HW existieren bereits				
Ebene	Hauptebene				
Hauptakteure	Kunde (dieser oder einer anderen Bank)				
Stakeholder u.	Kunde möchte schnell und sicher Geld erhalten				
Interessen	Bank möchte Personalaufwand reduzieren				
Voraussetzungen	Kunde hat gültige Karte				
	Bankomat läuft und hat Verbindung zum Banksystem				
Garantie	Es wird entweder (Geld ausgezahlt und abgebucht) oder (weder ausgezahlt noch abgebucht)				
Erfolgsfall	Der Kunde hat den gewünschten Geldbetrag erhalten; dieser Betrag wurde von seinem Kont				
Auslöser	Kunde steckt Karte in den Leser				
Beschreibung	1. Kunde steckt Karte in den Leser				
	2. System fragt nach PIN				
	3. Kunde gibt PIN ein				
	4. System fragt nach gewünschter Aktion				
	5. Kunde wählt "Abhebung" aus				
	6. System bietet Beträge an				
	7. Kunde gibt Wunschbetrag an				
	8. System prüft Verfügbarkeit und zahlt Betrag aus				
	9. System schließt die Sitzung, gibt Karte aus.				

Fortsetzung folgt ...

Kurt Schneider

Leibniz Universität Hannover


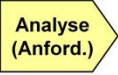
SWT 2015/16 - 279

Nur zur Erinnerung: Solche Tabelle beschreiben alles, was zu einem Use Case gehört. Die Struktur dient als Checkliste, damit man nichts vergisst. Außerdem können andere Entwickler, die dieselbe Tabelle verwenden, so sehr leicht verstehen, was ein konkreter Use Case bedeutet.

Dies ist der obere Teil der Tabelle.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover


		Use Case UC Geld abheben (Teil 2)					
Beschreibung	1.	Kunde steckt Karte in den Leser					
	2.	System fragt nach PIN					
	3.	Kunde gibt PIN ein					
	4.	System fragt nach gewünschter Aktion					
	5.	Kunde wählt "Abhebung" aus					
	6.	System bietet Beträge an					
	7.	Kunde gibt Wunschbetrag an					
	8.	System prüft Verfügbarkeit und zahlt Betrag aus					
	9.	System schließt die Sitzung, gibt Karte aus.					
Erweiterungen	3a	WENN keine gültige PIN eingegeben, DANN zurück zu 2 (nach drei Fehlversuche)					
	8a	WENN Wunschbetrag nicht verfügbar ist DANN					
	8a.1	bietet das System den höchsten verfügbaren Betrag an					
	8a.2	Kunde bestätigt, dass er ihn akzeptiert					
	8a.2a	Falls nicht akzeptiert, beende Sitzung (9).					
Technologie	8a.3	System zahlt Betrag aus, weiter bei 9					
		Bei Geräten mit Irissensor ist alternativ zur PIN-Eingabe auch ein Iris-Scan möglich					

... und das der untere.

Oft werden Use Cases in Excel oder anderen Tabellenformaten dargestellt.
Manchmal auch in Word, aber das ist unpraktischer.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

		Anforderungen in UC einbetten oder unter den Use Case schreiben	
UC 1	Geld abheben		
Umfeld	Bankautomat im Freien oder im Foyer		
Systemgrenzen	Kontoverwaltung und Bankomaten-HW existieren bereits		
Ebene	Hauptebene		
Hauptakteure	Kunde (dieser oder einer anderen Bank)		
Stakeholder u. Interessen	Kunde möchte schnell und sicher Geld erhalten Bank möchte Personalaufwand reduzieren		
Voraussetzungen	Kunde hat gültige Karte Bankomat läuft und hat Verbindung zum Banksystem		
Garantie	Es wird entweder (Geld ausgezahlt und abgebucht) oder (weder ausgezahlt noch abgebucht).		
Erfolgsfall	Der Kunde hat den gewünschten Geldbetrag erhalten; dieser Betrag wurde von seinem Konto abgebucht		
Auslöser	Kunde steckt Karte in den Leser		
Beschreibung	<div><div>1. Kunde steckt Karte in den Leser [R01] Phishing verhindern</div><div>2. System fragt nach PIN</div><div>3. Kunde gibt PIN ein [R02] PINs haben genau vier Stellen [R03] Alle vierstelligen Zahlen (0..9) sind als PIN erlaubt</div><div>4. System fragt nach gewünschter Aktion [R04] Es gibt zumindest "Abbruch" und "Abhebung"</div><div>5. Kunde wählt "Abhebung" aus</div><div>6. System bietet Beträge an [R05] Es wird angeboten: 50, 100, 150, 200, 300 EUR und "anderer Betrag"</div><div>7. Kunde gibt Wunschbetrag an</div><div>8. System prüft Verfügbarkeit und zahlt Betrag aus [R06] Als verfügbar gilt: Kontoguthaben plus Überziehungskredit</div><div>9. System schließt die Sitzung, gibt Karte aus.</div></div>		
Erweiterungen	3a WENN keine gültige PIN eingegeben, DANN zurück zu 2 (nach drei Fehlversuchen: Sperrung) [R07] Nach drei PIN-Fehlversuchen erfolgt Sperrung [R08] Fehlversuche auf Karte speichern und auch in Folgesession mitzählen		
	8a WENN Wunschbetrag nicht verfügbar ist DANN		
	8a.1 bietet das System den höchsten verfügbaren Betrag an [R09] Als verfügbar gilt: Kontoguthaben plus Überziehungskredit [R10] Wenn vorhandene Scheine nicht reichen, kürze ebenfalls		
	8a.2 Kunde bestätigt, dass er ihn akzeptiert		
	8a.2a: Falls nicht akzeptiert, beende Sitzung (9).		
	8a.3 System zahlt Betrag aus, weiter bei 9		
Kurt Schneider	Technologie	Bei Geräten mit Trissensor ist alternativ zur PIN-Eingabe auch ein Iris-Scan möglich	SWT 2015/16 - 281

Im nächsten Schritt werden die Use Cases weiter interpretiert und präzisiert. Hier sieht man, dass Anforderungen explizit an Stellen im Use Case geschrieben werden.

Wichtig ist die ID, z.B. [R02], der Kurztitel dient nur dem besseren Erinnern. Zu einer Anforderung gehören mehr Informationen, die man nicht alle in eine Use Case-Tabelle stecken kann oder will. Auch ist es sinnvoll, die Requirements im UC ausblenden zu können, damit man weiterhin den UC mit den Kunden besprechen kann.

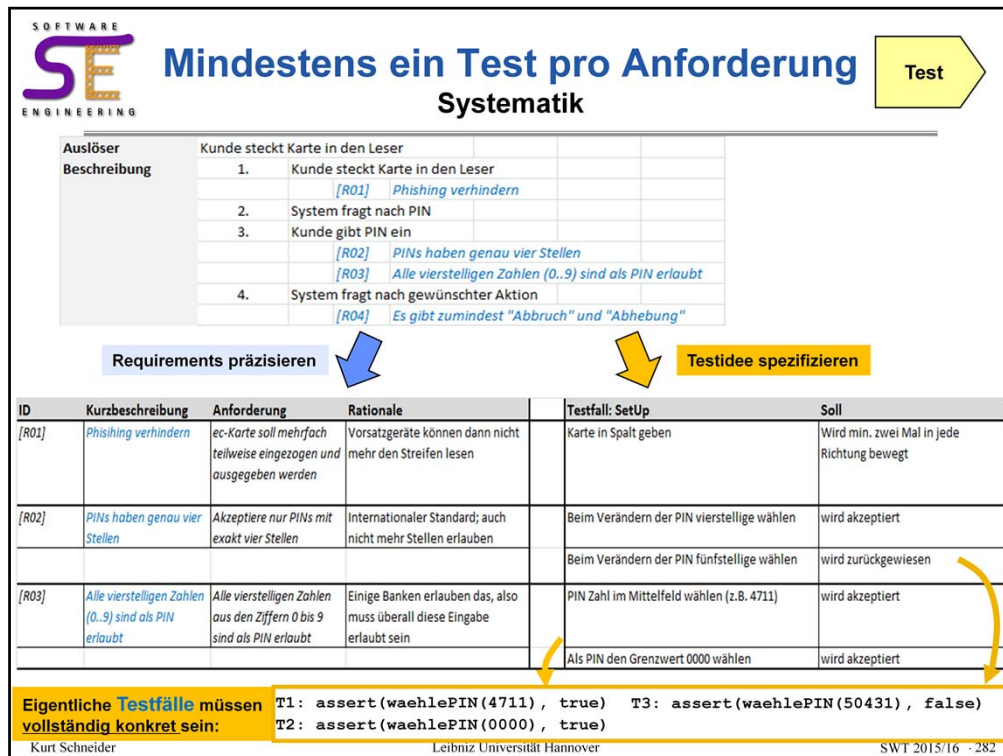
Wir betrachten nun an einem Ausschnitt (gelb), was mit den Anforderungen weiter geschieht.

Wo stecken überall Anforderungen?

- Im Prinzip überall im Use Case, also nicht nur in Beschreibung und Erweiterungen. Dort jedoch am meisten.
- Der dort beschriebene Ablauf muss vom System realisiert werden. Das führt alleine schon zu einigen Anforderungen.
- Weitere Detailanforderungen stehen bei den Schritten.
- Nicht vergessen:
Viele (Qualitäts-, Oberflächen-, aber auch andere funktionale) Anforderungen stecken nicht im Use Case. Es gibt also noch mehr Anforderungen. Siehe hierzu auch die Typen von Anforderungen in der Spezifikation.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier kommt es zu einer Aufspaltung in zwei weitere Wege, wie schon im Überblicksdiagramm gezeigt.

Die **konstruktive Seite der Entwicklung (blau)** führt zu konkreteren, besser verstandenen Anforderungen.

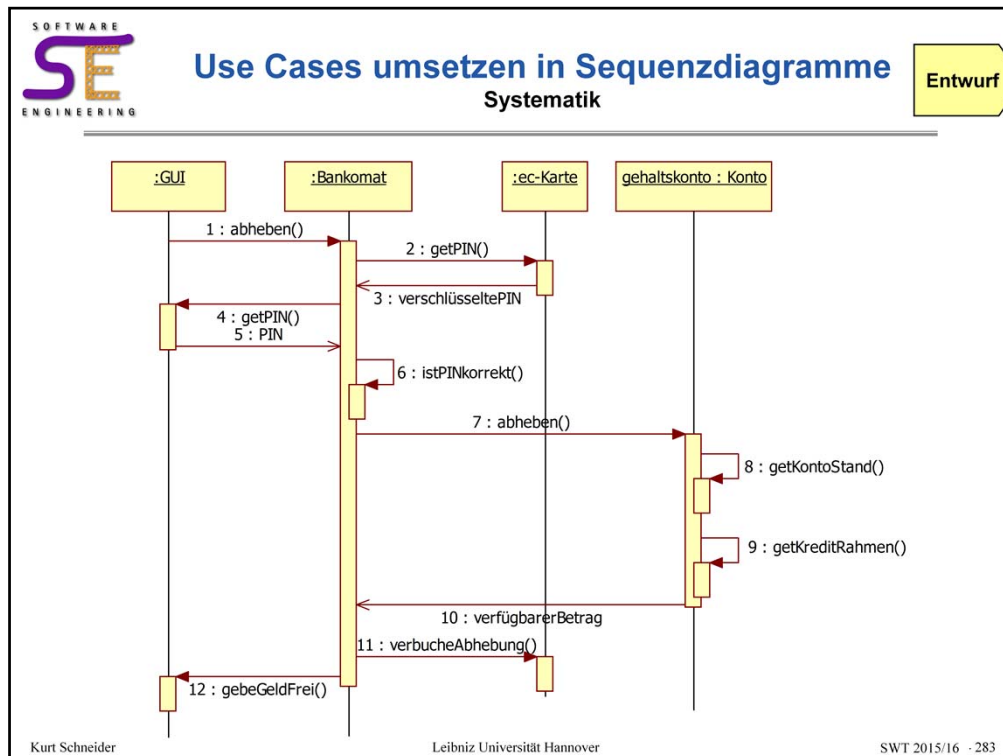
Die **Qualitätssicherungsseite (orange)** baut darauf auf: Es werden Testfälle spezifiziert. Darin wird noch nicht genau die Testfälle enthalten, die sind noch konkreter (unten). Aber immerhin beschreibt man schon sehr deutlich, wie man feststellen will, ob die Anforderung erfüllt ist.

Das hat zwei Vorteile:

- Man kann echte, vollständige Testfälle daraus entwickeln
- Und die Entwickler wissen noch genauer, was sie mit Ihren Programmen erreichen sollen. Die Testfälle bzw. -spezifikationen tragen also wesentlich zur Anforderungsklä rung bei.

Anforderungen und Testfälle sind idealerweise zwei Seiten einer Münze!

Eine Daumenregel heißt: Mindestens ein Testfall pro Anforderung – wie soll man sie sonst prüfen? Nebenbei führt das dazu, dass man schon sehr früh über Testen nachdenkt.



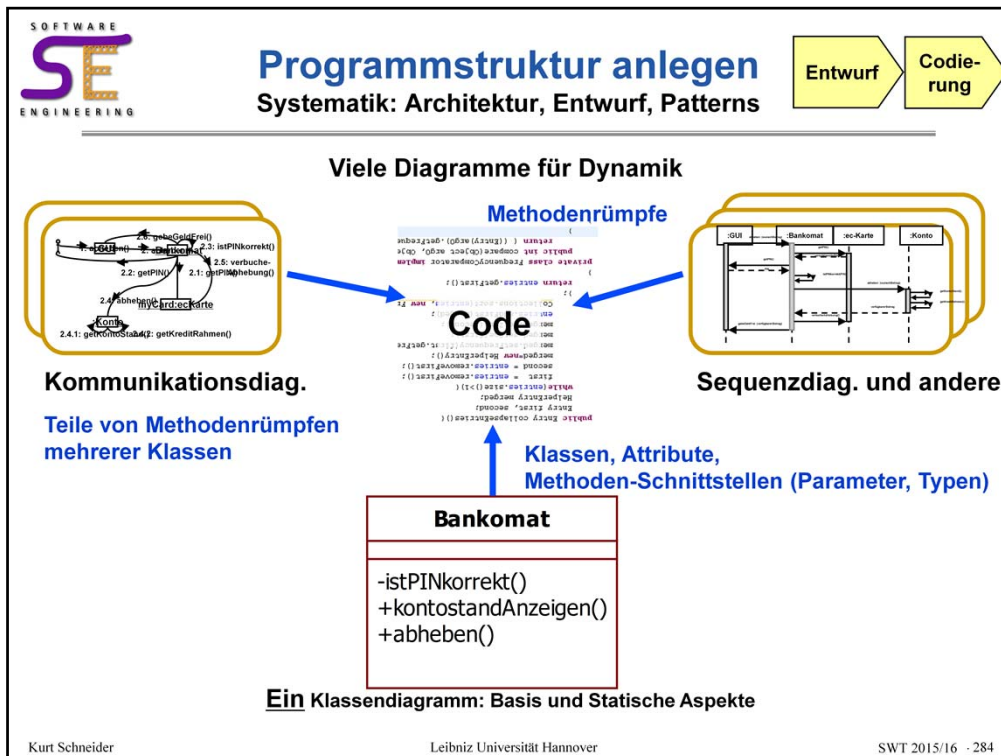
Dieses Sequenzdiagramm steht stellvertretend für den nächsten Schritt: Nun werden aus den Schritten Nachrichten abgeleitet, die verschiedene Objekte einander zuschicken.

Die Objekte hat man zuvor aus den Use Cases entnommen (die Akteure) oder aus der Objekt-Orientierten Analyse übernommen. Das Bild oben passt nicht genau zu den Use Cases (wurden unabhängig voneinander entwickelt); in einem realen Projekt würde man sich aber darum bemühen, den Übergang möglichst lückenlos zu gestalten.

Von hier aus kann man schon Coderahmen erzeugen lassen. Auch Entwickler wissen genau, wie sie aus UML-Modellen Programme ableiten können. Zum Beispiel kann man sich an der Lebenslinie eines Objekts entlangtasten, und daraus ablesen, welche Nachrichten das Objekt im Zusammenhang mit diesem Ablauf empfangen und verschicken können muss. Das kann man dann in der zugehörigen Klasse programmieren. Nun gibt es aber noch mehr Sequenz-, und vielleicht auch andere Diagramme. Wenn dort noch einmal ein Objekt derselben Klasse auftaucht, dann müssen auch die dortigen Aktivitäten in der Klasse programmiert werden. Das gesamte UML-Modell zusammen legt fest, was die verschiedenen Klassen können müssen, um alle Diagramm zu erfüllen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover




Diese Folie erinnert daran, dass zu einem UML-Modell immer ein Klassendiagramm und noch mehrere weitere UML-Diagramme gehören. In der Regel erhalten Entwickler diese Diagramme und können dann (wie vorhin kurz beschrieben) systematisch daraus Programme ableiten.

Inzwischen ist es oft möglich, aus den Diagrammen automatisch gewisse Teile des Programms zu erzeugen. Dann müssen nur noch die fehlenden Teile manuell ergänzt werden.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Guter Programmcode

Systematik

Codierung

Sie wählen im Quellcode:

- Klassennamen
- Methodennamen
- Leerzeichen
- Kommentare
- Einrückungen
- Variablentypen
- Schleifen
- Rückgabewerte
- ...

Ziele

- Verständlichkeit
- Lesbarkeit

```
// Collapse list of entries into a code tree. Return its r...  
  
private Entry collapseEntries(){  
    Entry first, second;  
    HelperEntry merged;  
  
    if(entries.isEmpty()){  
        return null;    // empty list, no tree can be built  
    }  
  
    while(entries.size()>1){  
        // Sort list by frequency, lowest frequency first  
        Collections.sort(entries, new FrequencyComparator());  
  
        first = entries.removeFirst();  
        second = entries.removeFirst();  
  
        // merge elements and put result back into list  
        merged=new HelperEntry();  
        merged.setFrequency(first.getFrequency()+ second.g...  
  
        // in Huffman codes, it does not matter who gets  
        merged.setOne(first);  
        merged.setZero(second);  
  
        // put result back (will be resorted if loop conti...  
        entries.addFirst(merged);  
    }  
};
```

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 -285

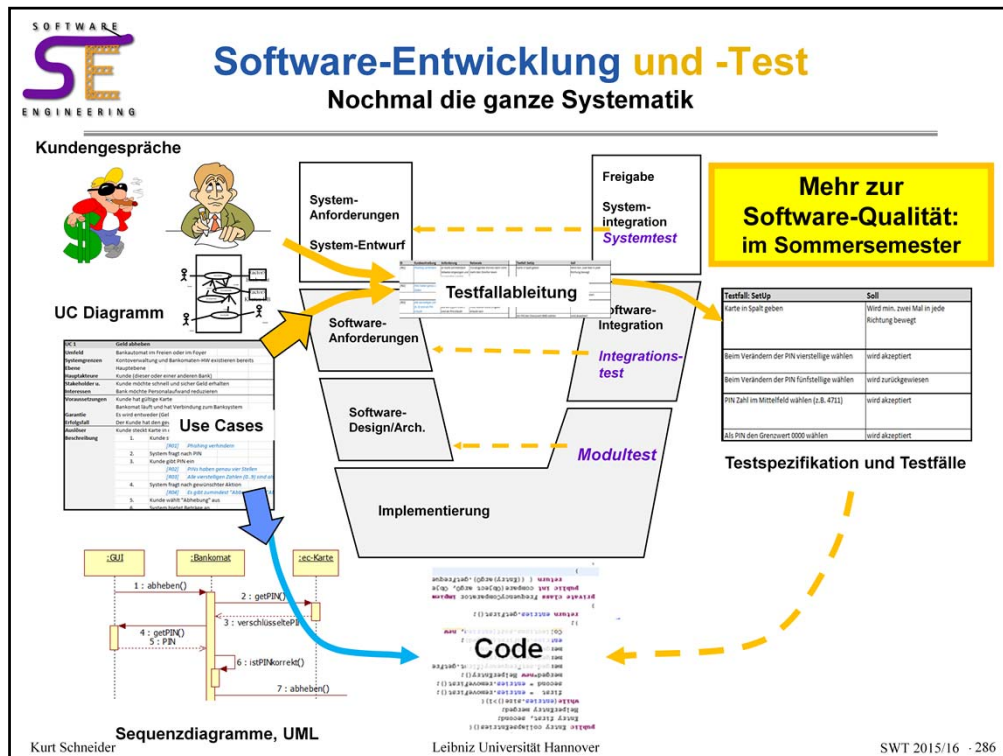
Egal, ob manuell oder generiert: Der entstehende Code sollte gut strukturiert und gut lesbar sein. Die obige Folie kennen Sie vom Anfang der Vorlesung. Sie macht deutlich, worauf sie bei der reinen Gestaltung des Programmtextes achten sollten.

Zusätzlich wird man aber auch bei der Struktur darauf achten, bewährte Muster einzusetzen: Beispielsweise die Drei-Schichten-Architektur, das MVC-Pattern und weitere Design Patterns.

Dies liegt in der Verantwortung und Kompetenz der Programmierer. Sie erhalten das UML-Modell als Vorgabe. Weitere Patterns, aber speziell auch Datenstrukturen und Algorithmen wählen die Entwickler passend zu den Qualitätsanforderungen und weiteren nicht-funktionalen Anforderungen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier noch einmal die Zusammenfassung auf einem Bild:

- Systematisch, also durchgängig aufeinander aufbauend, werden Anforderungen erfragt, geprüft und dokumentiert.
- Ein wesentlicher Teil der Anforderungen sind die funktionalen; sie bildet man auf Use Cases ab. Im UC-Diagramm sieht man eine Übersicht über alle vorhandenen UCs. Cases gehören also zur Anforderungs-Dokumentation.
- Dann hängt man Anforderungen in die Use Cases ein. Damit kann man sie einerseits präzisieren und konkretisieren; andererseits erhalten die Anforderungen damit auch einen Kontext (eben den UC), so dass man sie besser verstehen kann.
- Von nun an kann aus einem UC ein Sequenzdiagramm werden. Insgesamt entstehen also mehrere Sequenzdiagramme und ein ganzes UML-Modell.
- Das wird manuell oder automatisch unterstützt in Code umgesetzt. Dabei ist aber auch zu berücksichtigen, dass Architektur und Entwurf die geeigneten Design Patterns und Architekturen erhält.
- Inzwischen hat man sich von der Testseite herangearbeitet: Aus den präzisierten Anforderungen waren auch Testfallspezifikationen und schließlich Testfälle entstanden. Jetzt werden die Tests durchgeführt, von unten nach oben.
- Mit Hilfe der Tests findet man Fehler, korrigiert sie und erzeugt so systematisch hohe Software-Qualität.

In der SWQualitäts-Vorlesung im Sommer geht es damit weiter.