Prüfen objektorientierter Software

- ☐ Regeln für die Entwicklung
- ☐ Eigenschaften objektorientierter Systeme
- ☐ Objektorientierter Modultest: Klassentest
- ☐ Objektorientierter Integrationstest
- ☐ Objektorientierter Systemtest

Prüfen objektorientierter Software
Objektorientierung und Qualitätssicherung

+ Ansatzpunkt: Konzeptionelle Fehler
+ Verwendung von kommerziellen Bibliotheken (ausgetestet)
+ Wiederverwendung (Reuse)
+ Durchgängiges Konzept für Analyse, Entwurf, Implementierung
+ Klares Konzept (Regeln des Modells)
+ Paradigma für Problemanalyse
- Einziges Paradigma für Problemanalyse (Paradigma-Blindheit)

Prüfen objektorientierter Software
Objektorientierte Programmierung und Qualitätssicherung

+ Durchgängiges Konzept für Analyse, Entwurf, Implementierung
+ Hohe Produktivität (Bibliotheken)
+ Datenabstraktion
- Enorme Analyseprobleme beim Test (Dynamik ist statisch schwierig nachvollziehbar)
- Dynamisches Binden bereitet Probleme mit Echtzeit
- Polymorphismus
- Komplexitäten durch Vererbung

Entwickeln und Prüfen objektorientierter Software
Regeln für die Entwicklung: Analyse und Entwurf

- ☐ Modularisierung ist die Haupteigenschaft der Objektorientierung, die positiv auf die Prüfung wirkt:
  - Module sind klar identifiziert (Klassen).
  - Unabhängige Testbarkeit der Klassen aufgrund der Abgeschlossenheit
- ☐ Konsequenz: Modularisierungskonzept der Objektorientierung so nutzen, dass für die Prüfung der Software die Voraussetzungen erfüllt sind (muss spätestens beim Entwurf bedacht werden)
  - Modularisierungskonzept entsprechend der Objektorientierung konsequent durchhalten (z. B. Bilden von Datenabstraktionen durch Zusammenfassen logisch zusammengehöriger Daten und Funktionen in Klassen)
  - Abgeschlossenheitskonzept nicht durchbrechen (z. B. keine friend-Klassen in C++)
  - Keine Zugriffe auf Klassenattribute unter Umgehung der dafür zuständigen Methoden
  - Durch konsequente Anwendung der Verfeinerung OOA, OOD, OOP Konsistenz der Programmstruktur mit der Spezifikationsstruktur herstellen

Entwickeln und Prüfen objektorientierter Software
Regeln für die Entwicklung: Analyse und Entwurf

- Vererbung in Kombination mit Polymorphismus ist eine kritische Eigenschaft der Objektorientierung für den Test (Verständlichkeit der Struktur wird verringert)
  - Vererbung vorsichtig verwenden
  - Keine zu tiefen Vererbungshierarchien
  - Mehrfachvererbung nicht zu häufig verwenden
  - Konsequentes Durchhalten eines bestimmten Vererbungshierarchie (typischerweise vom allgemeinen zum speziellen)

Entwickeln und Prüfen objektorientierter Software
Regeln für die Entwicklung: Implementierung

- Beachten der oben erwähnten Regeln auch für die Implementierung
- Steigerung der Beobachtbarkeit der Klasseninterna (z. B. Werte der Klassenattribute) durch Verwendung von so genannten Zusicherungen (sind in C++ Bestandteil des Sprachumfangs)
- In komplexen oder kritischen Teilen der Software verständliche, einfache Programmierung gegenüber eleganten Lösungen vorziehen
- Dynamisches Binden in zeitkritischen Softwareteilen nicht nutzen

Prüfen objektorientierter Software
Eigenschaften objektorientierter Systeme

- Objekte und Klassen sind komplizierter als Funktionen.
- Untergrenze der Komplexität entspricht der von Datenabstraktionen bzw. abstrakten Datentypen in klassischen Softwareentwicklungen
- Objekte bzw. Klassen besitzen:
  - Beziehungen
  - Eigenschaften
  - Bestandteile bzw. sind selbst Bestandteil
  - einen Zustand.
- Sie können:
  - Botschaften versenden
  - Operationen ausführen.

Prüfen objektorientierter Software
Eigenschaften objektorientierter Systeme

- Sie besitzen:
  - Vorbedingungen
  - Nachbedingungen
  - Invarianten
  - Ausnahmebehandlungen (exceptions).
- Die Abläufe zwischen Teilen von Objekten sind kompliziert. => In objektorientierten Softwaresystemen sind wesentliche Teile des Komponententests identisch mit Integrationstestschritten für klassische Softwaresysteme.

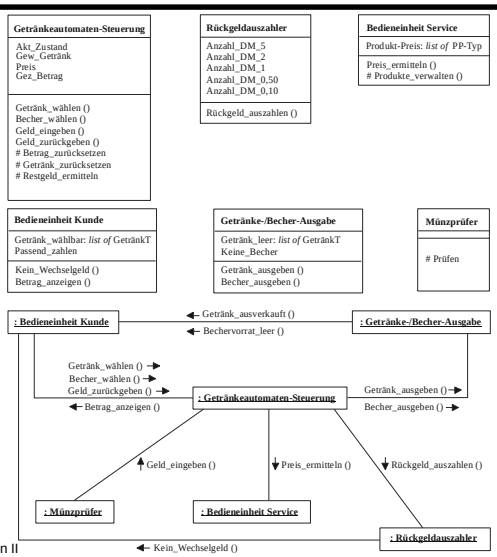
Prüfen objektorientierter Software
Objektarten

- Essentielle Objekte:
  - Modellieren Teile der Anwendung, werden als Teil der Systemanforderung identifiziert
- Nicht essentielle Objekte entstehen während späterer Phasen der Softwareentwicklung (Entwurf, Implementierung).
  - Sind Bestandteil der technischen Realisierung eines Softwaresystems
  - Sind oft Standardkomponenten

Prüfen objektorientierter Software
Objektorientiertes Beispiel

- Im Folgenden wird als Beispiel die Steuerung eines objektorientiert entworfenen Getränkeautomaten benutzt.
- Der Automat enthält als Klassen modelliert:
  - die Getränkeautomaten-Steuerung,
  - die Kundenbedieneinheit,
  - die Servicebedieneinheit,
  - den Münzprüfer,
  - die Getränke-/Becher-Ausgabe und
  - den Rückgeldauszahler.

Prüfen objektorientierter Software
Objektorientiertes Beispiel



Objektorientierter Modultest: Klassentest
Prüfung einzelner Operationen

- Das Prüfen von Methoden wird im Wesentlichen analog zum Modultest in der klassischen Softwareentwicklung durchgeführt.
- Aber Operationen
  - besitzen oft eine sehr einfache Kontrollstruktur,
  - sind stark abhängig von den Attributen des Objekts,
  - besitzen starke Abhängigkeiten untereinander.
- Die Prüfung einzelner Operationen (Funktionstest bzw. Strukturtest) ist sinnvoll unter folgenden Voraussetzungen:
  - Die Operation besitzt eine gewisse Mindestkomplexität.
  - Die Operation besitzt keine zu starken Abhängigkeiten zu anderen Teilen des Objektes
- Im Regelfall sind Operationen nicht sinnvoll einzeln prüfbar

Objektorientierter Modultest: Klassentest
Prüfung einzelner Operationen

- Nur im Ausnahmefall können Operationen allein geprüft werden; hier Spezifikation des Operation "Rückgeld_auszahlen":
- *Die Klasse "Rückgeldauszahler" enthält die Information über die für die Rückgeldauszahlung verfügbaren Münzen nach Art und Anzahl. Insgesamt sind max. 50 Münzen à 5 DM, 100 Münzen à 2 DM, 100 Münzen à 1 DM, 100 Münzen à 0,50 DM und 200 Münzen à 0,10 DM möglich. Rückgeld wird nach folgenden Regeln ausgezahlt:*
  - *Gezahlt wird mit der geringsten Anzahl Münzen, d.h. der Rückgeldbetrag wird zunächst ggf. mit 5 DM-Münzen beglichen, dann mit 2 DM-Münzen, anschließend 1 DM-Münzen, 0,50 DM-Münzen und 0,10 DM-Münzen. Falls eine benötigte Münzsorte nicht mehr verfügbar ist, so wird mit der nächstkleineren Sorte zurückgegeben.*
  - *Falls weniger als zwanzig 10-Pfennig-Münzen verfügbar sind, so wird die Botschaft Kein_Wechselgeld (ja) versandt, um die Passend zahlen-Anzeige zu aktivieren. Dies geschieht auch, wenn ein Gesamtbetrag des Wechselgelds mit Ausnahme der 5DM-Münzen von 5 DM unterschritten wird. Sonst wird die Botschaft Kein_Wechselgeld (nein) versandt.*

Objektorientierter Modultest: Klassentest
Funktionsorientierte Prüfung einzelner Operationen

- Funktionale Äquivalenzklassenaufstellung:

Bedingung	gültig		ungültig	
Rückgeld	Rückgeld ≥ 5,-	5,- > Rückg. ≥ 2,-	< 0	
	2,- > Rückg. ≥ 1,-	1,- > Rückg. ≥ 0,50		
	0,50 > Rückg. ≥ 0,10	0,10 > Rückg. ≥ 0		
Münzvorrat				
DM 5	50 ≥ Münzvorrat > 0	Münzvorrat = 0	< 0	> 50
DM 2	100 ≥ Münzvorrat > 0	Münzvorrat = 0	< 0	> 100
DM 1	100 ≥ Münzvorrat > 0	Münzvorrat = 0	< 0	> 100
DM 0,50	100 ≥ Münzvorrat > 0	Münzvorrat = 0	< 0	> 100
DM 0,10	200 ≥ Münzvorrat > 20	20 > Münzvorrat ≥ 0	< 0	> 200
Passend zahlen	Anzahl DM 0,10 < 20	Anz. 0,10 ≥ 20 und Gesamtbetrag ≥ 5,-		
	Gesamtbetrag < 5			

Objektorientierter Modultest: Klassentest
Funktionsorientierte Prüfung einzelner Operationen

☐ Testfälle für gültige Äquivalenzklassen:

Testfall-Nr.	1	2	3	4	5	6
Rückgeld	5 DM	2 DM	1 DM	0,50	0,10	0,00
Münzvorrat						
DM 5	50	1	0	10	0	10
DM 2	100	1	0	10	0	10
DM 1	100	1	0	10	0	10
DM 0,50	100	1	10	0	9	10
DM 0,10	200	20	0	19	4	40
Passend zahlen	Betr.≥ 5,- Anz. 0,10 ≥ 20	Betr.≥ 5,- Anz. 0,10 ≥ 20	Betr. =5,- Anz. 0,10 = 0	Betr.≥ 5,- 0 < Anz. 0,10 < 20	Betr. <5,- 0 < Anz. 0,10 < 20	Betr.≥ 5,- Anz. 0,10 ≥ 20
Ergebnis	1*5 DM, *Passend zahlen* nicht aktivieren	1*2 DM, *Passend zahlen* aktivieren	2*0,50, *Passend zahlen* aktivieren	5*0,10, *Passend zahlen aktivieren*	1*0,10, Passend zahlen aktivieren	*Passend zahlen* nicht aktivieren

Objektorientierter Modultest: Klassentest
Funktionsorientierte Prüfung einzelner Operationen

☐ Testfälle für ungültige Äquivalenzklassen:

Testfall-Nr.	7	8	9	10	11	12	13	14	15	16	17
Rückgeld	5,-	2,-	10,-	3,40	0,10	4,-	3,-	2,-	4,50	5,80	-0,10
Münzvorrat											
DM 5	51	10	20	10	10	-1	20	30	20	14	10
DM 2	10	101	20	10	10	10	-1	10	10	10	10
DM 1	10	1	101	10	10	10	30	-1	22	24	10
DM 0,50	10	1	10	101	9	10	40	2	-1	8	10
DM 0,10	30	42	30	40	201	10	50	49	30	-1	50

Objektorientierter Modultest: Klassentest
Funktionsorientierte Prüfung von Operationen im Kontext ihrer Klasse

- Im Regelfall müssen Operationen im Kontext ihrer Klasse geprüft werden
- Die Operationen eines Objekts einer Klasse stehen über gemeinsam genutzte Attribute in Wechselwirkung
- Die Werte der Attribute definieren den momentanen Zustand des Objekts

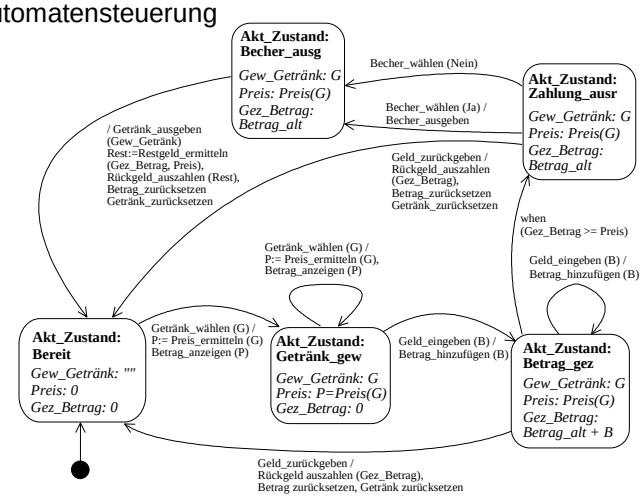
⇒ Zustandsautomat ist ein geeignetes Spezifikationshilfsmittel

⇒ Zustandsautomat kann als Basis der Prüfung dienen

- Anwendungsgebiet: Prüfung von Operationssequenzen

Objektorientierter Modultest: Klassentest
Funktionsorientierte Prüfung von Operationen im Kontext ihrer Klasse

- Beispiel: Getränkeautomatensteuerung



Objektorientierter Modultest: Klassentest Funktionsorientierte Prüfung von Operationen im Kontext ihrer Klasse

- Hierarchie von Vollständigkeitskriterien:
  - Mindestens einmalige Abdeckung aller Zustände
  - Mindestens einmaliges Durchlaufen aller Zustandsübergänge
  - Mindestens einmaliges Erzeugen aller Ereignisse an allen Zustandsübergängen
- Hierarchie:
  - Alle Ereignisse $\supseteq$ Alle Zustandsübergänge $\supseteq$ Alle Zustände
- Wichtig: Prüfung der Fehlerbehandlung nicht vergessen

Objektorientierter Modultest: Klassentest Funktionsorientierte Prüfung von Operationen im Kontext ihrer Klasse

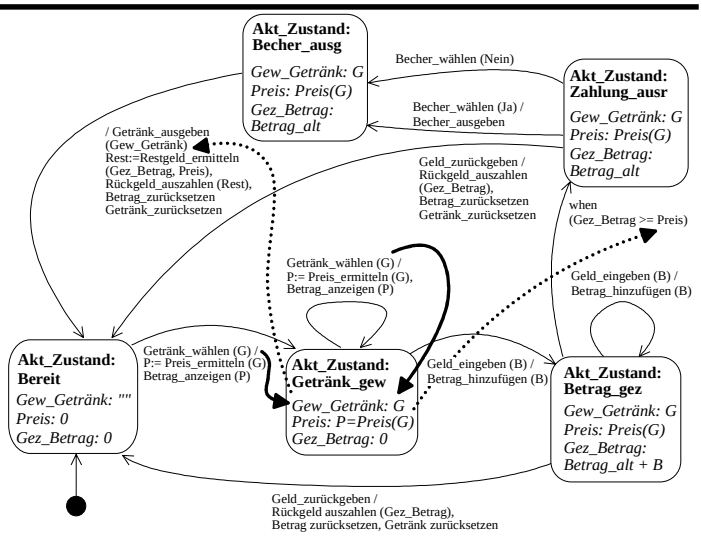
Der Test aller Zustandsübergänge ist z.B. mit den folgenden Testfällen möglich:

- Bereit, Getränk wählen -> Getränk gewählt, Getränk wählen -> Getränk gewählt, Geld eingeben -> Betrag gezahlt, Gezahlter Betrag < Preis des gewählten Getränks und Geld eingeben -> Betrag gezahlt, Geld zurückgeben -> Bereit
- Bereit, Getränk wählen -> Getränk gewählt, Geld eingeben -> Betrag gezahlt, Gezahlter Betrag >= Preis des gewählten Getränks -> Zahlung ausreichend, Geld zurückgeben -> Bereit
- Bereit, Getränk wählen -> Getränk gewählt, Geld eingeben -> Betrag gezahlt, Gezahlter Betrag >= Preis des gewählten Getränks -> Zahlung ausreichend, Becher wählen(Ja) -> Becher ausgeben, - -> Bereit
- Bereit, Getränk wählen -> Getränk gewählt, Geld eingeben -> Betrag gezahlt, Gezahlter Betrag >= Preis des gewählten Getränks -> Zahlung ausreichend, Becher wählen(Nein) -> Becher ausgeben, - -> Bereit
- *Die Bestimmung der Werte für Schnittstellenparameter kann durch Äquivalenzklassenanalyse geschehen.*

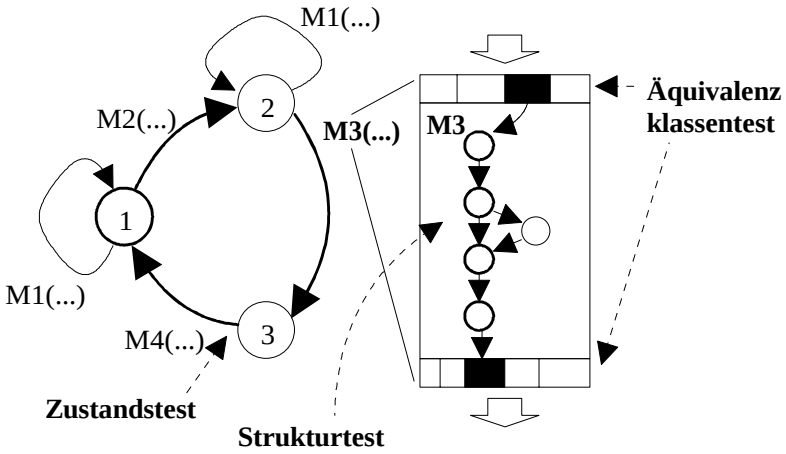
Objektorientierter Modultest: Klassentest
Strukturorientierte Prüfung

- Kontrollflußorientierte Testtechniken (z.B. Zweigüberdeckungstest) sind relativ ungeeignet, weil sie die Kopplungen zwischen Operationen eines Objekts durch gemeinsam genutzte Attribute nicht beachten
- Datenflußorientierte Testtechniken sind besser geeignet, weil sie diese Kopplungen beachten
- Die Attribute werden von Operationen geschrieben (def) und gelesen (use). Ein Datenflußtest auf Basis der Attribute fordert den Test von Interaktionen über die gemeinsam genutzten Daten
  (Beispiel: Definition und Benutzung der Attribute "Gew_Getränk" und "Preis" im Zustand "Getränk_gew" der Getränkeautomatensteuerung -> nächste Seite)

Objektorientierter Modultest: Klassentest
Strukturorientierte Prüfung



Objektorientierter Modultest: Klassentest
Sinnvolle Testvorgehensweise



Objektorientierter Modultest: Klassentest
Prüfung "generischer" Klassen

- Problem:
  - Abstrakte und parametrisierte Klassen gestatten keine (direkte) Erzeugung von Objekten
  - Vielfalt der erzeugbaren Objekte erhöht die Komplexität
  - Getestet werden können nur konkrete Objekte. Frage: Welche?
  - Abstrakte und parametrisierte Klassen verhalten sich zu konkreten Klassen wie normale Klassen zu ihren Objekten.
- Abstrakte Klassen
  - Legen die syntaktische und semantische Schnittstelle für Operationen fest, ohne eine Implementation anzubieten
  - Die Implementation für abstrakte Methoden wird in einer Unterklasse der abstrakten Klasse gegeben.
  - Strukturieren eine Menge von Klassen
- Parametrisierte Klassen
  - Enthalten formale Klassenparameter, die durch aktuelle Werte (Klassen) ersetzt werden müssen.

Objektorientierter Modultest: Klassentest
Testen abstrakter und parametrisierter Klassen

- Instanzierung einer konkreten Klasse
- Test dieser Klasse wie eine gewöhnliche Klasse
- Fragen:
  - Welche Instanzierung ist zu wählen?
  - Wie ist beim Testen methodisch vorzugehen?
- Regel: Erzeugung einer möglichst einfachen konkreten Klasse, d. h.:
  - Abstrakte Klassen:
    - Realisierung von Implementationen für abstrakte Methoden
      - Falls möglich, leere Implementationen

      Sonst:
      - So einfach wie möglich, mit der Nebenbedingung, dass die Spezifikation erfüllt wird; nur so kompliziert wie erforderlich, um einen sinnvollen Test zu gewährleisten.
  - Parametrisierte Klassen:
    - Wahl von Parametern, die den Test möglichst einfach gestalten (z. B. "Stapel für Integer")

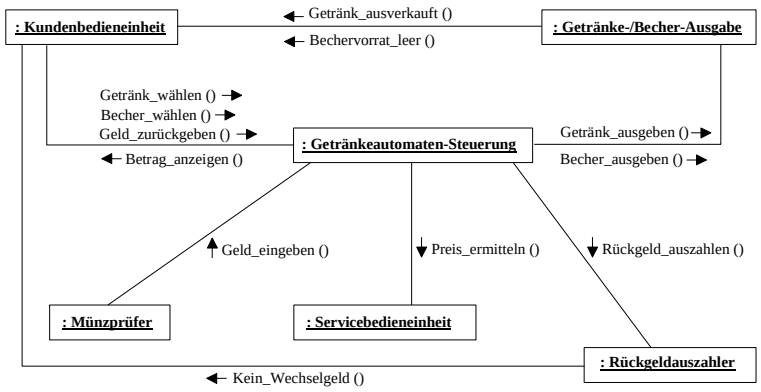
Objektorientierter Integrationstest
Integrationstest von Basisklassen

- Voraussetzung: Getestete Basisklassen aus dem Modultest
- Fragen:
  - Sind die Aufrufe von Diensten des Dienstanbieters seitens des Dienstbenutzers korrekt?
  - Funktioniert die Übergabe und Interpretation von Ergebnissen korrekt?
- Idee:
  - Überdeckung der Spezifikation des Dienstnutzers und Dienstanbieters
- Erzeugung von Testfällen,
  - die die unterschiedlichen vom Dienstanbieter erzeugbaren Rückgabewerte überdecken.
  - die die unterschiedlichen vom Dienstnutzer verarbeitbaren Rückgabewerte überdecken

=> Funktionale Äquivalenzklassenbildung der Schnittstelle zwischen Dienstbenutzer und Dienstanbieter

Objektorientierter Integrationstest
Integrationstest von Basisklassen

- Beispiel: Integrationstest der Klassen "Münzprüfer" und "Getränke-automaten-Steuerung"



Objektorientierter Integrationstest
Integrationstest von Basisklassen

Test der Interaktion der Klassen "Münzprüfer" und "Getränkeautomaten-Steuerung" über die Botschaft "Geld_eingeben ()"

- Schnittstellenspezifikation der Operation "Geld_eingeben ()":
  - Die Operation "Geld_eingeben ()" erwartet einen nicht-negativen Schnittstellenparameter, der maximal 1000 sein kann. Er gibt den Geldbetrag in Pfennigen an.
  - Die Operation besitzt keinen Rückgabewert
- Schnittstellenspezifikation der Operation "Prüfen ()" in Bezug auf die Botschaft "Geld_eingeben ()":
  - "Prüfen ()" belegt den Schnittstellenparameter der Botschaft "Geld_eingeben ()" mit einem der folgenden Werte: 10, 50, 100, 200, 500.

Objektorientierter Integrationstest
Integrationstest von Basisklassen

- Folgen für den Integrationstest:
  - Es ist sicherzustellen, dass der Betrag, der der Operation "Geld_eingeben (Betrag)" übergeben wird, die folgende Bedingung erfüllt (sog. Zusicherung): *(Betrag >= 0) AND (Betrag <= 1000)*
  - Äquivalenzklassen und gleichzeitig Testfälle für den Aufruf (Schnittstellen-Äquivalenzklassen des Dienstbenutzers in Aufrufrichtung):
    - Betrag = 10
    - Betrag = 50
    - Betrag = 100
    - Betrag = 200
    - Betrag = 500
  - Test der zurückgelieferten Ergebnisse nicht erforderlich, da keine Rückgabewerte vorhanden

Objektorientierter Integrationstest
Integrationstest in Gegenwart von Vererbung

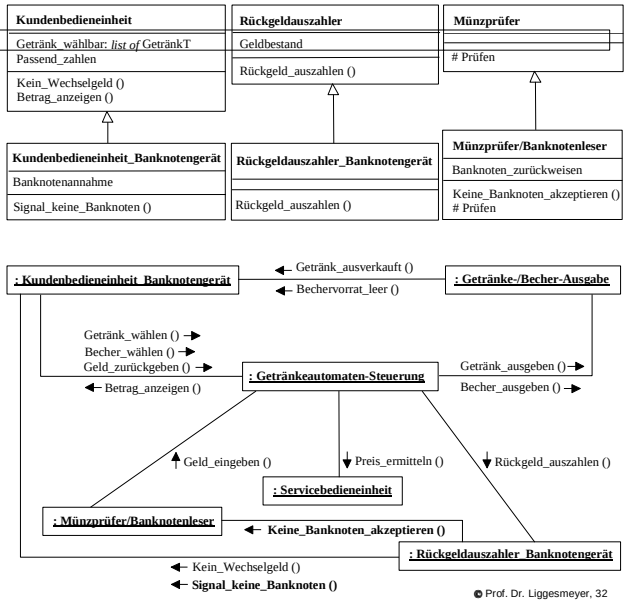
- Unterschiedliche Situationen:
  - Integrationstest von dienstanbietenden abgeleiteten Klassen
  - Integrationstest von Dienstaufrufen aus abgeleiteten Klassen
  - Integrationstest dienstanbietender und dienstnutzender Unterklassen

Objektorientierter Integrationstest
Integrationstest in Gegenwart von Vererbung

- Beispiel
  In der neuen Version des Getränkeautomaten ist es möglich, mit Banknoten (10 DM und 20 DM) zu zahlen. Um diese Funktionalität zu realisieren, werden folgende Änderungen vorgenommen:
  - Zu der Klasse "Münzprüfer" wird durch Vererbung eine neue Klasse "Münzprüfer/Banknotenleser" erzeugt, deren Methode "Prüfen()" um das Prüfen der Banknoten erweitert ist. Diese Methode überschreibt die ursprüngliche Methode. Ferner wird eine neue Methode "Keine_Banknoten_akzeptieren ()" vereinbart, deren Ausführung je nach Parameter die Akzeptanz von Banknoten sperrt oder freigibt.
  - Zu der Klasse "Rückgeldauszahler" wird eine Unterklasse vereinbart, da eine die alte Methode überschreibende Methode "Rückgeld_auszahlen ()" erforderlich ist, die den Klassen "Münzprüfer/Banknotenleser" und "Kundenbedieneinheit_ Banknotengerät" signalisiert, ob mit Banknoten gezahlt werden kann. Das Zahlen mit Banknoten wird gesperrt, falls der Geldbestand in Münzen 30 DM unterschreitet. Banknoten werden als Wechselgeld nicht zurückgegeben.
  - Zu der Klasse "Kundenbedieneinheit" wird eine Unterklasse erzeugt, die um die Methode "Signal_keine_Banknoten ()" erweitert ist. Diese Methode setzt ein Signal, das dem Kunden die Freigabe bzw. Sperrung der Zahlung mit Banknoten signalisiert.

Objektorientierter Integrationstest
Integrationstest in Gegenwart von Vererbung

- Der Getränkeautomat mit Banknotenleser



Objektorientierter Integrationstest: Vererbung
Integrationstest von dienstanbietenden abgeleiteten Klassen

- Situation:
  - Der Dienstbenutzer benutzt Dienste einer abgeleiteten Klasse.
- Voraussetzung:
  - Dienstnutzer, Dienstanbieter und Oberklasse des Dienstanbieters sind sinnvoll klassengetestet.
  - Integrationstest des Dienstnutzers und der Oberklasse des Dienstanbieters entsprechend der Vorgehensweise zum Integrationstest von Basisklassen durchgeführt
- Problem:
  - Operationen des Dienstanbieters können von der Oberklasse (dem alten Dienstanbieter) unverändert geerbt sein, müssen aber nicht. Es können sowohl Methoden des neuen Dienstanbieters als auch Methoden des alten Dienstanbieters (der Oberklasse) ausgeführt werden.
- Fragen:
  - Sind die Aufrufe von Diensten seitens des Dienstbenutzers korrekt?
  - Funktioniert die Übergabe und Interpretation von Ergebnissen korrekt?

Integrationstest von dienstanbietenden abgeleiteten Klassen
Korrektheit der Aufrufe von Methoden des Dienstanbieter

Vorgehensweise:

- Keine zusätzlichen Testfälle für ererbte Methoden, da dieser Fall bereits durch den Integrationstest der Basisklassen abgedeckt ist => Testfälle wiederholen
- Keine zusätzlichen Testfälle bei überschriebenen Methoden, für die sich lediglich die Implementation geändert hat, da die Schnittstellenspezifikation identisch geblieben ist und dieser Fall ebenfalls bereits abgedeckt ist => Testfälle wiederholen
- Falls sich die Schnittstellenspezifikation der überschreibenden Methode geändert hat, so ist eine Fallunterscheidung nötig:
  - Die Schnittstelle der überschreibenden Methode ist spezieller (d. h. akzeptiert weniger Daten) als die Schnittstelle der überschriebenen Methode.
    - Definition einer neuen Zusicherung

Integrationstest von dienstanbietenden abgeleiteten Klassen Korrektheit der Aufrufe von Methoden des Dienstanbieters

- Wiederholung sämtlicher Testfälle aus dem Integrationstest der Basisklassen; Reaktionsmöglichkeiten im Fehlerfall:
  - Modifikation des Dienstbenutzers, so dass nur korrekte Aufrufe abgesetzt werden
  - Modifikation des Dienstbenutzers, so dass keine Aufrufe zur überschreibenden Methode abgesetzt werden
  - Modifikation der überschreibenden Methode, so dass alle Aufrufe des Dienstbenutzers korrekt verarbeitet werden können

- Falls die Schnittstelle durch das Überschreiben der Methode allgemeiner wird, so sind keine zusätzlichen Testfälle erforderlich, da dieser Fall bereits durch den Test der Basisklassen abgedeckt wird => Testfälle wiederholen

Integrationstest von dienstanbietenden abgeleiteten Klassen Korrekte Übergabe und Interpretation der Ergebnisse

- ☐ Ggf. zusätzliche Testfälle für die Überdeckung einer breiteren Schnittstellenspezifikation der überschreibenden Methode, die beim Test der Basisklassen nicht hinreichend abgedeckt worden sind
- ☐ Beispiel: Getränkeautomat mit Banknotenleser
  - Der durch Vererbung entstandene neue Rückgeldauszahler ist ein Dienstanbieter ("Rückgeld_auszahlen ()") gegenüber der Getränkeautomatensteuerung.
  - Die überschreibende Methode "Rückgeld_auszahlen ()" besitzt im Vergleich zur überschriebenen Methode jedoch nur eine geänderte Implementation (Versand zusätzlicher Botschaften). Die Schnittstellenspezifikation ist unverändert.
  - Für den Integrationstest von "Getränkeautomatensteuerung" und "Rückgeldauszahler_Banknotengerät" ist es ausreichend, die alten Testfälle zu wiederholen. Die Zusicherung ist unverändert.

Integrationstest von Dienstaufrufen aus abgeleiteten Klassen Überprüfung der Korrektheit der Aufrufe

- Keine zusätzlichen Testfälle für nicht überschreibende Methoden des Dienstnutzers => Testfälle wiederholen
- Keine zusätzlichen Testfälle, falls die Schnittstelle der überschreibenden Methode in Aufrufrichtung spezieller ist als die Schnittstelle der überschriebenen Methode (d. h. Aufrufe, die vorher möglich waren, sind nicht mehr möglich) => Testfälle wiederholen
- Falls die Schnittstelle durch das Überschreiben der Methode allgemeiner wird (d. h. Aufrufe, die vorher nicht möglich waren, sind möglich), so sind die alten Testfälle entsprechend zu ergänzen => alte Testfälle wiederholen und neue ergänzend durchführen.
- Anmerkung: Die Zusicherung ist unverändert

Integrationstest von Dienstaufrufen aus abgeleiteten Klassen Test der korrekten Interpretation von Übergabeparametern

- Testfälle wiederholen. Falls ein Fehler aufgrund einer spezielleren Schnittstelle in Rückgaberichtung auftritt, so ist eine entsprechende Korrektur erforderlich.

Integrationstest dienstanbietender und dienstnutzender Unterklassen

- Vorgehensweise:
  - Integrationstest der dienstanbietenden Unterklasse
  - Integrationstest der dienstnutzenden Unterklasse
  - Zusätzliche Testfälle für die Wechselwirkung in dienstanbietender und dienstnutzender Unterklasse

Integrationstest dienstanbietender und dienstnutzender Unterklassen

Beispiel: Getränkeautomat mit Banknotenleser

- Zwischen den abgeleiteten Klassen "Rückgeldauszahler_Banknoten-gerät" und "Münzprüfer/Banknotenleser" besteht eine Dienstanbieter-Dienstnutzer-Beziehung. Zusätzlich zu den beschriebenen Tests ist diese durch Überprüfung der Interaktion durch die Botschaft "Keine_ Banknoten_akzeptieren ()" zu testen.
- Testfälle
  - Keine_Banknoten_akzeptieren (ja)
  - Keine_Banknoten_akzeptieren (nein)
- Eine analoge Situation existiert zwischen "Rückgeldauszahler_ Banknotengerät" und "Kundenbedieneinheit_Banknotengerät".

Objektorientierter Integrationstest
Vererbung und Integrationstest: Zusammenfassung

☐ Basistabelle für die Beachtung von Vererbung

Dienstnutzer	Dienstanbieter	Aktion
unverändert	unverändert	Testfälle wiederholen
unverändert	durch Vererbung erzeugt	Tabellen 1.1 und 1.2 auswerten
durch Vererbung erzeugt	unverändert	Tabelle 2 auswerten
durch Vererbung erzeugt	durch Vererbung erzeugt	Tabellen 1.1, 1.2 und 2 auswerten; Testfälle für Interaktion der Subklassen ergänzen

Objektorientierter Integrationstest
Vererbung und Integrationstest: Zusammenfassung

☐ Tabelle 1.1: Test der Aufrufschnittstelle

Dienstanbietende Operation	Aufrufschnittstelle der dienstanbieten-den Operation	Aktion
geerbt	-	Testfälle wiederholen
überschreibend	identisch	Testfälle wiederholen
überschreibend	spezieller	Neue Zusicherung; Testfälle wiederholen
überschreibend	allgemeiner	Testfälle wiederholen

Objektorientierter Integrationstest
Vererbung und Integrationstest: Zusammenfassung

☐ Tabelle 1.2: Test der Rückgabeschnittstelle

Dienstanbietende Operation	**Rückgabeschnitt-stelle der dienstan-bietenden Operation**	**Aktion**
geerbt	-	Testfälle wiederholen
überschreibend	identisch	Testfälle wiederholen
überschreibend	spezieller	Testfälle wiederholen
überschreibend	allgemeiner	Testfälle wiederholen; zusätzliche Testfälle erzeugen

Objektorientierter Integrationstest
Vererbung und Integrationstest: Zusammenfassung

☐ Tabelle 2: Test der Aufruf- und der Rückgabeschnittstelle

Dienstanbietende Operation	**Aufrufschnittstelle der dienstanbieten-den Operation**	**Aktion**
geerbt	-	Testfälle wiederholen
überschreibend	identisch	Testfälle wiederholen
überschreibend	spezieller	Testfälle wiederholen
überschreibend	allgemeiner	Testfälle wiederholen; zusätzliche Testfälle erzeugen

Objektorientierter Systemtest

- Mit Ausnahme des Funktionstests keine Unterschiede gegenüber dem Test von klassischer Software:
  - Das System ist eine Black Box => Ob es objektorientiert oder klassisch entwickelt ist, spielt keine Rolle
- Erzeugung funktionsorientierter Testfälle aus OOA-Diagrammen:
  - DFDs
  - Zustandsautomaten (Rumbaugh)
  - Use Cases nach Jacobson:
    - Szenarien aus der Sicht eines Systembenutzers (Mensch oder anderes System)
    - Nicht sehr systematisch
    - Keine Vollständigkeit bei komplizierten Systemen
    - Können mit Zeitbedingungen annotiert werden => Leistungstest!