Themengebiete „Softwaretechnik“

- **Grundlagen**
  - Definitione
  - Formale Sprache
  - Schnittstelle
- **Anforderungen an den Softwareentwicklungsprozess**
- **Prozessmodelle**
  - Wasserfallmodell von Royce (1970)
  - V-Modell
  - Evolutionäres Modell
    - Explorative Programmieren
    - Throwaway Prototyping
  - Unified Process Modell von Rational
  - Spiralenmodell von Boehm
  - Wichtige Attribute eines Prozessmodells
- **Anforderungsanalyse**
  - Bedarfsanalyse
  - Bedarfsspezifikation
  - Software Spezifikation
  - Strukturierung von Anforderungen
  - Projekt Skizze
    - Qualitätskriterien
  - Bedarfsplan
- **Versionsmanagement**
  - Konfigurationsmanagement
  - CVS
- **Modellierung**
  - Modellierungssprachen
    - E/R-Diagramm
    - Datenflussdiagramm
    - Klassenmodell
    - UML
    - Use Case Diagramm
      - Sequence Diagramm
    - Class Diagramm
      - Klassen
    - CRC Karten
    - Dynamische Modellierung
      - State Charts
        - Hierarchien
        - Parallel Vorgänge
        - Default Connector
        - History Connector
        - Switch
      - Activity Diagrams
    - UML: Sequenzdiagramme
- **Formale Sprachen - Z**
  - Z – Spezifikation
    - Ausdrücke
    - Prädikate
  - Getypte Mengentheorie
    - Äquivalenz von Mengen
    - Per Extension definierte Mengen
    - Potenzmenge
    - Kartesisches Produkt
    - Axiome
  - Typen in Z
    - Basistypen
    - Freie Typen
    - Produkttypen
    - Typechecking

    - Objekte in Z
    - Relation => Dom , Range
    - Z-Spezifikationen
        - Überschreiben von Relationen
        - Projektion
        - Inverse von Relationen
        - Kombination von Relationen
        - Kommutativität
        - Iterationen in Kompositionen
        - Transitive und reflexive Hülle
    - Funktionen
        - Partielle-, totale Funktionen
        - Lambda Notation (Abkürzung)
        - Verschiedene Abbildungen
    - Mengen
        - Endlichkeit
        - Zahlenbereiche
        - Kardinalität
        - Sequenzen
            - Mengen von Sequenzen
        - Multimengen
    - Z-Schemata
        - Horizontale vs. Vertikale Syntax
        - Tagged Record Types
        - Charakteristische Bindung
        - E/R Diagramm nach Z umwandeln
        - Äquivalenz von Schemata
        - Z-Schemata als Prädikate
        - Auflösen von Constraints im Deklarationsteil
        - Logische Operationen auf Z-Schemata
            - Umbenennung
            - Konjunktion
            - Inklusion
            - Decoration
            - Disjunktion
            - Verhindern ungültiger Eingaben
            - Negation von Schemata
            - Quantifikation (Allquantor) und Hiding (Existenzquantor)
            - Komposition (Piping)
- **Formale Sprachen – Haskell**
    - Datentypen, Abstrakter Datentyp
- **Formale Sprachen – SML**
    - Strukurierungsmöglichkeiten
    - Kapselung
    - Interface
    - 2 Arten von Vererbung
- **Vom Entwurf zum Code**
    - **Vorteile von Klassendiagrammen**
    - **Begriff der Verfeinerung**
    - **Abbildung von Z auf Code**
        - **Case Tools**
        - **Design Patterns**
- **GUIs**
    - Designziele
    - WIMP-Style Design
    - Systemintegrationsgüte
    - Formale Software validierung
        - CSP
    - Java
        - AWT/SWING
- **SmlTK**

- **Middleware**
  - CORBA
    - IDL
    - ORB
    - Statische vs. Dynamische Aufrufe
- **Softwarevalidierung**
  - 3 Dimensionen
    - Validierungstechniken
    - Validierungsstufen
    - Ziele der Validierung
  - Post-Verification Approach
  - Development by Refinement
  - Transformational Approach
  - Codeinspektion
- **Testen von Software**
  - Testmethoden
    - Akzeptanztest
    - Integrationstest
    - Unittest
  - Fault Based Testing
  - Specification Based Input Space Partitioning
  - Uniformitäts Hypothese
  - Regularitätshypothese
  - DNF Methode
  - Algebraische Methode

Softwaretechnik Fragenkatalog:

1. **Wozu wird Softwaretechnik benötigt?** [Vermeidung von Fehlern durch Analysen (z.B. Einfahren des Fahrwerks auf der Landebahn, Geschwindigkeit <60 >60 aber was bei =60| Softwarekrise 1965| 70-80% angefangener Projekte werden fertig gestellt| 50-60 Fehlerzeilen auf 1000 Zeilen Code, 4-5 Fehlerzeilen auf 1000 bei Fertigstellung| Viele Anwendugen erfordern Korrektheit um jeden Preis, jedoch gibt es kein „silver bullet"]
2. **Was ist Softwaretechnik (Definition) ?** [Softwaretechnik ist die praktische Anwendung von wissenschaftlichen Methoden zu Spezifikation, Entwurf und Erstellung von Programmen und Systemen.]
3. **Wann wird eine Sprache formal genannt?** [… wenn für sie eine formale Sprache (Syntax) festgelegt ist, deren Bedeutung (Semantik) mathematisch genau beschrieben ist.]
4. **Wann wird eine Entwicklungsmethode formal genannt?** [… wenn sie auf einer formalen Sprache basiert und mit der Semantik konsistente Umformungs- und Beweisregeln vorliegen]
5. **Welche semi-formale Sprache kennen Sie?** [UML]
6. **Welche Vorteile haben formale Techniken?** [Vergleichsweise geringer Umfang| Präzision| Genaue Umformungsregeln → maschinelle Verarbeitung möglich| Einheitlicher Rahmen für Spezifikation, Entwickung, Testen, …| Nachteil: schwieriger für Laien]
7. **Wie bewältig man die hohe Komplexität in der Programmentwicklung?** [Durch Strukturierung und Abstraktion| Strukturierung dient der Gliederung der Aufgabenstellung| Abstraktion: Elemination unwichtiger Details]
8. **Welche klassischen Strukturierungs- und Vereinfachungsmittel kennen Sie?** [Funktionale Dekompostion: Zerlegung in unabhängige, in sich geschlossene Teilaufgaben| Parametrisierter und generischer Entwurf: Wiederverwendbarkeit| Einfache Denkmodelle: Besseres Verständnis der Aufgaben und möglicher Lösungen| Abstraktionsebenen: Ebenen unterschiedlicher Detailierungsgrade| Information-Hiding: Schnittstellen und eigenschaftsorientierte Beschreibung]
9. **Was ist eine Schnittstelle?** [Gedachter oder tatsächlicher Übergang an der Grenze zwischen 2 Funktionseinheiten mit den vereinbarten Regeln für die Übergabe von Daten oder Signalen| Die Schnittstelle ist auch die Verständigungsgrundlage zwischen dem Spezifizierer, Implementierer und Benutzer| Die korrekte Beschreibung und Benutzung einer Schnittstelle ist ein zentraler Aspekt der Softwaretechnik]
10. **Nennen Sie die traditionelle Methode ein Programm zu schreiben und zu verbessern!** [Code and Fix ➔ Funtkioniert wunderbar in 1-Mann Projekten bei kleineren Projekte]
    a. **Welche Probleme treten besonder beim Code-and-Fix Design auf?** [Wartbarkeit und zuverlässigkeit nimmt zunehmend ab. Wenn der Programmierer kündigt ist alles vorbei| Total ungeeignet für Riesenprojekte mit mehreren Personen-Jahren| Wenn der Entwickler und der End-Benutzer unterschiedliche Personen sind unterscheiden sich die Erwartungen gewaltig]
11. **Welche Anforderungen sollte der Entwicklungsprozess für ein gutes Softwareprojekt erfüllen?** [Es sollte eine angemessene Prozedur geben, die den Entwicklungsprozess leitet und kontrolliert| Es sollte die Entwicklung von High-Quality Systemen unterstützen]
    a. **Welche Attribute kommen dabei besonders zum tragen?** [Adequatheit: System erfüllt gewünschte Anforderungen| Verwendbarkeit (Usability): Passende Benutzerschnittstellen (GUIs), Dokumentationen etc.|Zuverlässigkeit (Reliabilität) Korrektheit, Sicherheit, Robustheit| Wartungsfähigkeit: System soll einfach zu verbessern und zu modifizieren sein| Kosten: Entwicklungskosten (d.h. Zeit und Geld) sollen im Rahme bleiben| Leistung: Ressourcenbedarf minimieren bzw. nicht verschwenden]
12. **Welche vier Hauptschritte begleiten den Softwareentwicklungsprozess?** [Spezifikationsphase: Definition der Funktionalität und der (Neben-) bedingungen| Entwicklung und Implementierung des Produktes| Validierungsphase; Verifikation und Testen| Wartungsphase: Verbesserungen und zukünftige Anpassungen]
13. **Nennen Sie Beispiele für Prozessmodelle!** [Wasserfallmodell| Evolutionäres Modell| Formale Transformation| Aufbau von wieder verwendbaren Komponenten (Bibliotheken ➔ JEDAS)]
    a. **Erklären Sie das Wasserfallmodell von Royce (1970) genauer!** [Der Entwicklungsprozess wird in Phasen zerlegt. Eine Phase muss abgeschlossen sein, bevor die nächste beginnt. Jede Phase produziert ein Produkt, d.h. ein Dokument

oder ein Programm| Das Ergebnis jeder Phase ist die Vorraussetzung (Eingabe) der darauffolgenden Phase]

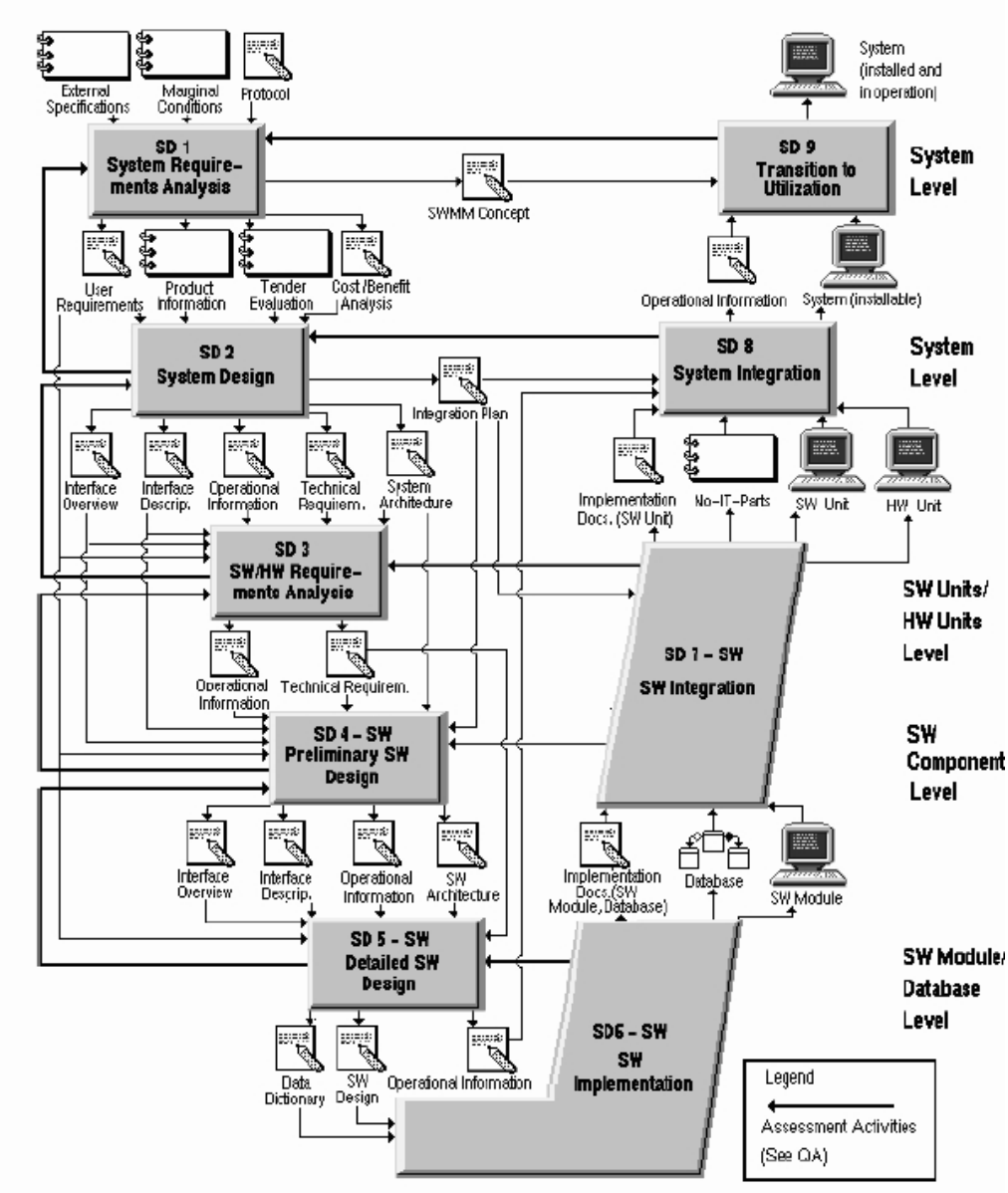
i. **Welches sind diese 5 Phasen?**
    1. Phase: Bedarfsanalyse und Bedarfsdefinition: Anwendung wird mit dem Kunden besprochen. Es wird so definiert, dass es für beide Seiten (also Kunden und Entwickler) verständlich ist]
    2. Phase: Systementwurf: Bedarf ist gegliedert in Hardware und Softwaresysteme. Systemarchitektur ist festgelegt.
    3. Phase: Implementation und Prüfung von Einheiten: System ist als Menge der Einheiten realisiert. Jede Einheit wird getestet.
    4. Phase: Integration und Systemtesting: Einheiten werden dabei zusammengeführt und geprüft, danach dem Kunden geliefert.
    5. Phase: Operation und Wartung: Fehler beseitigen (Bugfixes), System verbessern etc.

ii. **Welches sind die großen Vorteile des Wasserfallmodells?** Bedarf und Anforderungen sind von Anfang an bekannt. Diese Anforderungen ändern sich fast nicht mehr. Der Designprozess kann in abstrakter Weise geleitet. Alles passt schön zusammen am Ende ☺]

iii. **Welches sind die Probleme die beim Wasserfallmodell auftreten?** [Die Anforderungen sind typischerweise am Anfang sehr unpräzise und entwickeln sich erst währen des Entwicklungsfortschrittes| Big Bang Theorie ist riskant, d.h. Erst am Schluss zu schauen was tatsächlich herauskommt!| Viel zu viel Dokumentation| Der späte Einsatz verbirgt viele Risiken, z.B.: Technologischer Irrtum ➔ „Ich dachte die arbeiten zusammen| Konzeptueller Irrtum ➔ „Ich dachte die wollten das so“| Personelle Fehlplanung ➔ „hat so lange gedauert, dass die Hälfte des Teams abgesprungen ist“| Der Endbenutzer sieht das Ergebnis erst am Schluss und findet es grundsätzlich erstmal schlecht ➔ frustrierend für das Team| Testphase kommt viel zu spät| Unidirektionaler Entwicklungs-Fluss ist zu starr➔Feedback zwischen den Phase wird eigentlich benötigt ➔ Probleme werden in der Folge weitergegeben oder umschifft]
    1. **Welche Probleme bringt das hinzufügen eines iterativen Feedbacks in die Wasserfallphasen?** [Es ist schwer tatsächliche Checkpoints zu setzen]

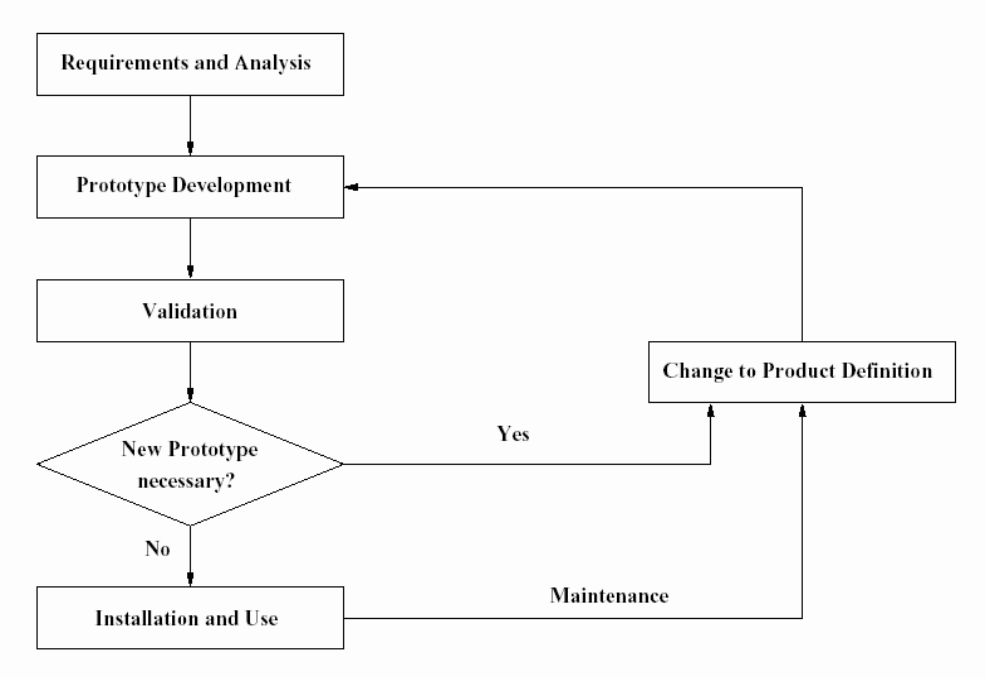
b. **Nennen und erklären Sie eine Variante des Wasserfallmodells!** [V-Modell: Ist ein ISO-Standard für Militärprojekte und teilweise Bundesverwaltungsprojekte.| Es reguliert alle Aktivitäten und Produkte, sowie Produktzustände und Zusammenhänge während der IT-Entwicklung und –Wartung. Es besteht aus unterschiedlichen Teilmodell. Es beschreibt Systementwicklung, Konfigurationsverwaltung, Projektverwaltung (Beschaffung, Planung, …), usw.]

i. **Wie sieht das V-Modell aus?**
    [



]

c. **Wie funktioniert das evolutionäre Modell?** [Basiert auf der Idee, einen Prototypen zu entwickeln und mit Hilfe der Kunden immer wieder zu verbessern ➔ Alle Prozessaktivitäten passieren gleichzeitig]
    i. **Es gibt dennoch zwei verschiedene Varianten, wie unterscheiden sich diese?** [Exploratives Programmieren: Befragt den Bedarf der Kunden. Man fängt zunächst mit dem Grundbedarf für das System an und fügt neue Möglichkeiten später hinzu.|Throw-Away-Prototyping: Das Ziel ist es durch verschiedene Prototypen den Bedarf des Kunden zu verstehen. Mit Hilfe des Prototyps experimentiert man mit dem Kunden um den Bedarf genauer zu definieren. Der Prototyp wird dann ggf. modifiziert bzw. neu entwickelt solange bis der Kunde zufrieden ist und das Produkt annimmt

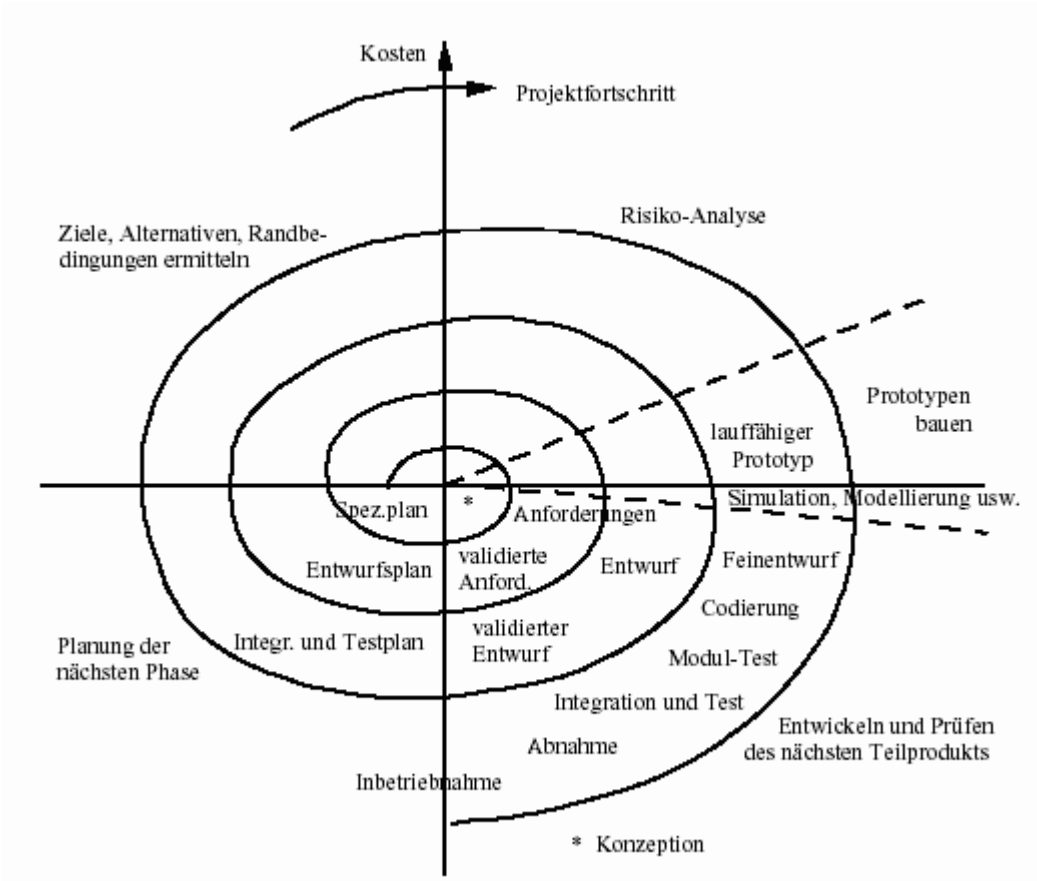


]

ii. **Nennen Sie Vor- und Nachteile des evolutionären Modells!** [Theoretisch sehr weit anwendbar➔ Viele System sind auf diese Art entwickelt worden, aber: Der Prozessverlauf ist nicht transparent genug, d.h. die Manager haben keine Checkpoints, man weiss nicht wo man steht. Schlechte Strukturierung aufgrund der ständigen Veränderunngen ➔ Erfordert viel können kleiner, motivierter und geschickter Teams. Praktisch gesehen ist die Anwendbarkeit also klein, nur für kleine System mit kürzerer Lebensdauer.]

d. **Wie funktioniert das Unified Process Modell der Firma Rational?** [Kombination aus Wasserfall und evolutionärem Modell| Versucht das Fehlerrisiko zu minimieren➔ Das System wird in kleine Teil-Projekte aufgeteilt, die riskanteren Teile werden genauer und als erstes betrachtet. Die Teilprojekte werden via Wasserfallmodell eingebettet in UML bearbeitet was ein frühe Einbeziehung aller am Gesamtprozess Beteiligten (Tester, Implementierer, Dokumentierer) zur Folge hat.

e. **Wie funktioniert das Spiralenmodell von Boehm?** [Umfasst evolutionäre Aspekte und Risikobewertung. Jeder Umlauf der spirale entspricht dem nächsten Prototypen. Winkel entspricht Zeit und Radius entspricht Kosten Jeder Umlauf zerfällt in 4 Hauptphasen



Das Wasserfallmodell kann auch eingebettet werden➔ Analyse und Definition bzw. Entwurf tauchen als eigene Windungen auf. Implementierung und Test werden in einem Zyklus zusammengefasst]

f. **Welche Attribute sind Relevant um ein Prozessmodell zu bewerten?**
   i. **Verständlichkeit?** [Wie genau ist der Prozess definiert, wie einfach ist er zu verstehen]
   ii. **Transparenz?** [Liefern die Akitvitäten klar definierte Ergebnisse?]
   iii. **Unterstützung?** [Können die Aktivitäten maschinell unterstützt erden]
   iv. **Annehmbarkeit?** [Wird der Prozess von den Entwicklern werden?]
   v. **Zuverlässigkeit?** [Werden Fehler vermieden oder schnell entdeckt]
   vi. **Robustheit?** [Kann der Prozess fortgesetzt werden auch wenn Probleme auftreten?]
   vii. **Wartung?** [Wie einfach können Änderungen integriert werden?]
   viii. **Zeitfaktor?** [Wie schnell kann ein System gebaut werden?]

14. **Was unterscheidet Software-Entwicklung z.B. mit dem Bau eines Schiffes?** [Software ist nicht greifbar|Softwareentwicklung und Softwareprodukte sind nicht standardisiert➔ Der Bau von Schiffen, die Art der Schiffe und die Überprüfung sind schon weitgehend standardisiert bzw. beruhen auf einer bewährten Tradition| Software-Fehler werden aber geduldet ein sinkendes Schiff hingegen nicht]
15. **Was ist das Ziel der Anforderungsanalyse?** [Lege die Anforderungen so genau wie möglich aber auch so abstrakt wie möglich dar.]
    a. **Weshalb benötigt man diese Anforderungsanalyse?** [Um zu verstehen, wie das Produkt eigentlich funktionieren soll, ansonsten braucht man gar nicht anfangen zu programmieren| Aufgrund der Anforderungsanalyse wird der Vertrag mit dem Kunden geschlossen| Die Entwicklung wird mit der AA geplant. Viele Projekte erfordern eine konkrete Produktdefinition ➔ Stichwort: Pflichtenheft]
16. **Warum ist gerade die Planung so schwierig?** [Es ist schwer herauszufinden was man will! Kein anderer Teil der Planungsphase ist so schwer, denn wenn man hier einen Fehler macht ist dieser im Nachhinein am schwersten zu korrigieren| Des Weiteren ist es schwierig Kompromisse zwischen widersprüchlichen Benutzeranforderungen zu finden Beispiel: Bankkunden wollen Sicherheit ihrer Daten, die Bank hingegen will möglichst wenig

Overhead und Beschränkungen in der Handhabung. Langzeitwirkungen bestimmter Effekte sind fast nicht abschätzbar z.B. Zweckentfremdung bestimmter Funktionen]

17. **Wie könnte man die Anforderungsanalyse weiter strukturieren?** [Man teilt diese so auf, dass die unterschiedlichen Beteiligten auf jeweils Ihrer Weise daran teilnehmen können.]
    a. **Bedarfsanalyse (Produktskizze)?** [an ihr nehmen Kunden und Benutzer auf der einen Seite und Vertragsmanager und Systemarchitekten auf der anderen Seite Teil. Das Problem wird grob erläutert als Produktskizze und in natürlicher Sprache und mit Darstellungen formuliert und dargestellt.]
    b. **Bedarfsspezifikation (Produktdefinition)?** [An ihr nehmen Kunden, Systemarchitekten und auch schon die Programmieren Teil. Das Problem wird detailiert erläuter und per genauer Produkt Definition (Pflichtenheft) in ein präzises und strukturiertes Dokument gepackt welches als Vertrag dient]
    c. **Software Spezifikation?** [Hier sind nur noch die Systemarchitekten und die Programmierer beteiligt. Eine grobe Lösung wird gesucht und diese wird in einer Systemarchitektur modelliert]
18. **Warum strukturiert man Anforderungen?** [Eins ist klar, unstrukturierter Text ist schlecht, weil verschiedene Anforderungen in einem (unstrukturierten) Satz gemischt werden können. Die Beschreibung ist u.U. unvollständig. Spezifikationen und Begründungen (Mutmaßungen) werden möglicherweise gemischt.]
19. **Wie strukturiert man die Anforderungen?** [Beschreibung fundamentaler Datenstrukturen und Operationen, dann werden die Hauptoperationen beschrieben➔ Eine Standardstruktur gibt es nicht ➔ Verschiedene Organisationen haben eigen Standards ➔ Aber es gibt einige allgemeine Eigenschaften: z.B. Beschreibung von Zielen, Funktionalität und Umgebung/Nutzung ➔ Die Beschreibung sollte abstrakt sein und unnötige Verpflichtungen sollten vermieden werden]
20. **Welches sind die wichtigsten Bestandteile einer Projekt-Skizze?** [7 Stück: Abschnitt 1: Problembeschreibung und Projektziele| Abschnitt 2:Funktionsumfang und Aussenverhalten (Welche Funktionen werden alle benötigt)| Abschnitt 3: Benutzerprofil (Welche Benutzer nutzen das System wie?| Abschnitt 4: Akzeptanz Kriterien (Was muss unbedingt erfüllt sein)| Abschnitt 5: Entwicklungs- Einsatz- und Wartungsumgebung, Schnittstellen, Nebenbedingungen (Betriebssystem, Rechnerumgebung, Netzwerkart, GUI-Art, Datentransfer zusätzlich Garantiebedingungen etc.)| Abschnitt 6: Lösungsstrategien, Generelle Architektur (Wie wird das umgesetzt?)| Abschnitt 7: Informationsquellen➔ Ansprechpartner, Manuals und Glossar]
21. **Welche Qualitätskriterien muss die Projekt-Skizze am Ende erfüllen?** [Gültigkeit (Validität) d.h. Ist die richtige Funktionalität spezifiziert und für wen? |Konsistenz: Die Anforderungen sollen nicht miteinander in Konflikt stehen| Vollständigkeit: Die gesamte Funktionalität soll spezifiziert werden| Realistisch: Das Projekt muss unter den gegebene finanziellen, zeitlichen und auch allen anderen Bedingungen erfüllbar/implementierbar sein.]
22. **Wie überpüft man die Projekt-Skizze auf ihre Qualitätskriterien?** [Walk-Through, Gespräche, „Requirements Review“ Prototyp, Logik Basierte Werkzeuge]
23. **Wie sieht ein Bedarfsplan aus?** [Eine noch detailiertere Produktskizze, Kombination aus nicht Formaler (natürlicher) Sprache und Semiformaler Sprache (Graphische Notation, Entwurfsbeschreibungssprache)| Selten eine ganze Formale Sprachen (wie Automaten, Petrinetze, Logik erster Ordnung)➔ Weniger Kundenfreundlich)]
24. **Wie kann man den Bedarfsplan z.B. graphisch modellieren?** [z.B. Datenflussmodellierugn: Beschreibt die Daten die eingegeben werden, was mit diesen geschieht und was ausgegeben wird| z.B. Ereignis orientierte Modellierung: System besteht aus Zuständen und reagiert auf Ereignisse, ähnlich wie ein Automat aber mächtiger, z.B. Hierarchisch oder mit parallelismus, Vollformal, Basis für Synchron oder Echtzeitprogrammiersprachen, Funktionale Anforderungen sind gar nicht formalisiert]
25. **Was bedeutet Versionsmanagement?** [Verwaltung von Abfolgen (verschiedene Versionen) von Dokumenten und deren Rekonstruktion. Optimierung und Kompression| Verwaltung Arbeitsteiliger Entwicklung und Synchronisation (Merge)| Generierung von Release]
    a. **Welche Konzepte kommen dabei zum Einsatz?** [Verschiedene Versionszweige, Verschiedene Module, Gemeinsamer Daten-/Dokumentenpool (Repository), Verteilung der Daten/Arbeit, Sicherheitskonzepte]
    b. **Wie wird auf den einzelnen Dokumenten gearbeitet, Direkt im Repository oder auf lokalen Kopien und warum?** [Lokale Kopien, denn es können ja mehrere Leute das gleiche Dokument bearbeiten. Beim Commit werden die gemeinsam veränderten Teile eines Dokumentes gemerged]

c. **Wann entstehen unterschiedliche Entwicklungszweige?** [z.B. Wenn eine bestimmte Version eines Programms in zwei oder mehrere Richtungen entwickelt wird um beispielsweise eine besser Lösung zu finden| oder z.B. wenn im Nachhinein eine Bugfix für eine bereits Realeaste Version herauskommen muss]
d. **Welche beiden Ansätze gibt es Konflikte zu vermeiden?** [Ressource Locking, d.h. ein gerade bearbeitetes Dokument wird für weitere Schreibzugriffe / Arbeitskopien gesperrt (ausgecheckt)| Merging: Zwei gleichzeitig bearbeitete Versionen werden gemerged (CVS)]
e. **Welche beiden Szenarien gibt es beim CVS beim mergen?** [Geänderte Textstellen überschneiden sich nicht ➔ Mering ist einfach| Textstellen überschneiden sich ➔ Nacheditiierung notwendig]
f. **Was versteht man unter Tagging beim Versionsmanagement?** [Symbolische Namen für Mengen von Revisionen (Konfiguration)➔ d.h. verschiedene Versionen einzelner Files stellen ein bestimmtes „Tag" dar, d.h. eine Konfiguration, dabei muss nicht immer die neueste Version eines Files im „Tag" sein.]
g. **Was ist Konfigurationsmanagement?** [Verwaltung verschiedener Konfigurationen (Mächtiger als reine Versionsverwaltung) erfordert mitverwaltetes Datenabhängikeitsmodell – Änderungen müssen bekannt gemacht und verwaltet werden. Kontrolle des gesamten Entwicklungsprozesses, Workflowmanagement, bekannte Systeme sind z.B. Aegis, Clearcase, StarTeam]
h. **Vorteile gegenüber reinem Versionsmanagement?** [Bestmögliche Rekonstruierbarkeit, Effektives Management verschiedenster Konfigurationen möglich, File-System-Struktur kann versioniert werden, Datenabhängigkeitsmodell kann versioniert werden.]
i. **Was ist CVS (Concurrent Version System)?** [Schwaches Konfigurationsmanagementsystem (eher Versionmanagement) ➔ Entferntes gemeinsames Repository| Mehrfache Arbeitskopien ohne sperren| Version heisst hier eher Revision| Open-Source| Unix-Add-On| Verschiedene Konfigurationen können registriert werden]
j. **Welche Hauptbereiche deckt das CVS System ab?** [CVS verwaltet: den Erstellungsprozess, d.h. Arbeitskopien, Files/Directorys, Projekte/Module und das Repository| es verwaltet Differenzen, Veränderungen, Löschungen, Konfigurationsbildungen, Status| CVS sorgt für die richtige Synchronisierung, d.h. Arbeitskopien vs. Repository, Modulen und Verzweigungen| Als letztes hat es allgemeinen Aufgaben wie Benachrichtigungen der Teilnehmer, Logfiles, Geeignete Zusammenfassung aller Daten]
k. **Welche Schwächen hat CVS?** [Keine einfache Bedienung (Komplexe Befehle)| Eigentliche Standardoption ist oft schwer zu lösen| Rechtevergabe von Unix ist schwer| Keine Kontrolle von Datenabhängigkeiten, d.h. das System muss monoton wachsen]
l. **Wohin geht die Reise?** [Versionsmanagement ist eine der Schlüsseltechnologien der Softwaretechnik, aber es muss weiter gehen hin zum Konfigurationsmanagement. Weitere Forschungsgebiete sind auf jeden Fall verschiedene Mergeoperationen bezogen auf reinen Text, Quellcode etc.

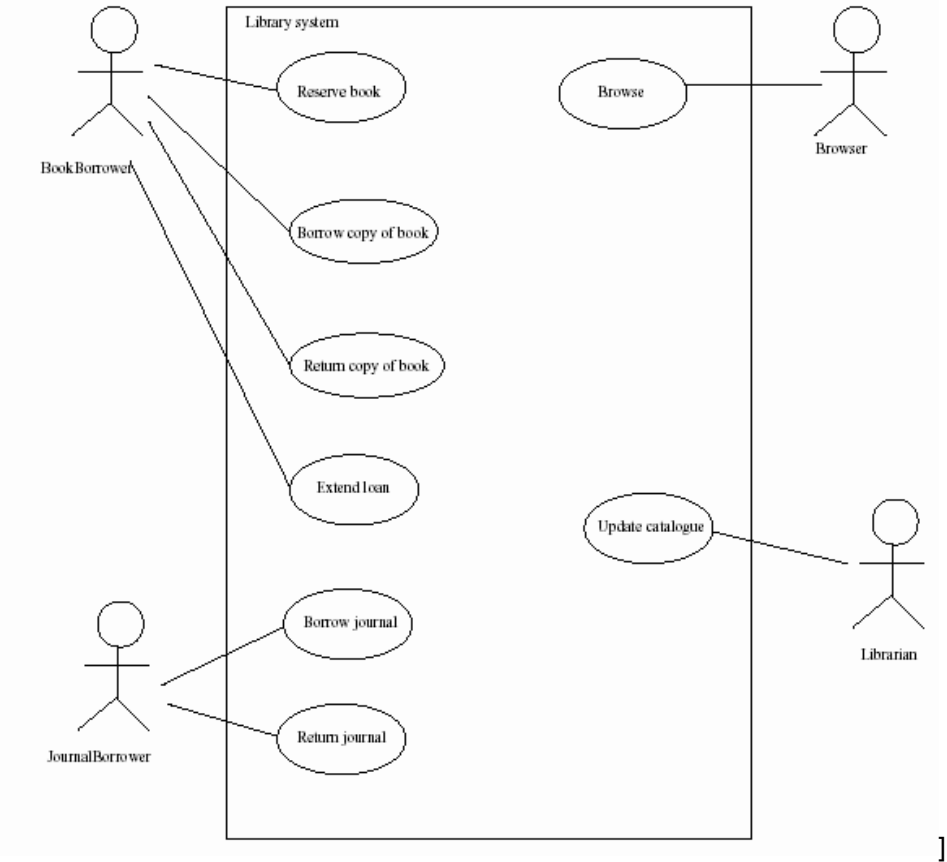
26. **Was ist ein Modell?** [Ein Modell ist ein Konstrukt oder mathematisches Objekt, welches ein System beschreibt. Oder es stellt eine Theorie dar, die die Eigenschaften eines Systems beschreibt (s=v*t ist ein Modell).In der Informatik modellieren wir Systeme und ihre Einsatzumgebung, um deren Verhalten zu klären und Eigenschaften zu analysieren.]
27. **Wie unterscheiden sich verschiedene Modellierungssprachen?** [Sicht des Systems, z.B. Statisch, dynamisch, funktional, OO, …|Abstraktionseben, z.B. Anorderungen vs. System Architektur|Grad der Formalität, d.h. Informal, semiformal oder formal| Ausprägungen (Religion), z.B. OO-school, Oxford Z/CSP, Church of HOL]
28. **Wie sind die einzelnen Modellierungssprachen entstanden?** [Sie sind alle eine eine Mixtur aus verschiedenen anderen Sprachkomponenten wie z.B. E/R Diagramme, Flusdiagramm, Klassendiagramme etc.]
29. **Was ist ein E/R Diagramm?** [Spezifiziert Mengen gleichartiger Daten und ihre Beziehungen (Relationen)| Es gibt 3 arten von Objekten die visuell spezifiziert sind: Entitäten, Attribute und Relationen]
30. **Was ist ein Datenflussdiagramm?** [Graphische spezifikationssprache für Funktionen und Datenfluss| Es gibt Symbole für Funktionen, Eingabe, Ausgabe, Datenfluss und Datenspeicherung| Leider nicht standardisiert]

31. **Was ist ein Klassenmodell?** [Betont Objekt und Zusammenhänge| System wird mit Hilfe von Objekten beschrieben, die an Hand ihrer Attribute (und Funktionen) in Klassen zusammengefasst werden. Das Ziel ist die Verständlichkeit und Wiederverwendbarkeit| Klassenhierarchie drückt über Vererbung gemeinsame Eigenschaften und Methoden aus]
32. **Wie ist UML historisch gesehen entstanden?** [Seit den 70er Jahren gab es mehrere konkurrierende OO-Modellierungsmethoden von denen sich 3 (OOD,OMT und OOSE) etablierten und von den 3 Entwicklern neben weiteren Sprachkomponenten in UML standardisiert werden (ist noch im Gange)]
33. **Was ist UML?** [Eine Kombination aus 9 Modellierungssprachen die unterschiedliche Aspekte von System auf unterschiedlichen Abstraktionsebenen modellieren können, da gibt es z.B. das „Use Case Model" welches das System aus der Benutzersicht beschreibt| Das „Static Model", welches Elemente des Systems und ihre Beziehungen beschreibt| „Dynamic Modell"- Welches das [Zeit]- verhalten des Systems beschreibt]

Major Area	View	Diagrams	Main Concepts
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

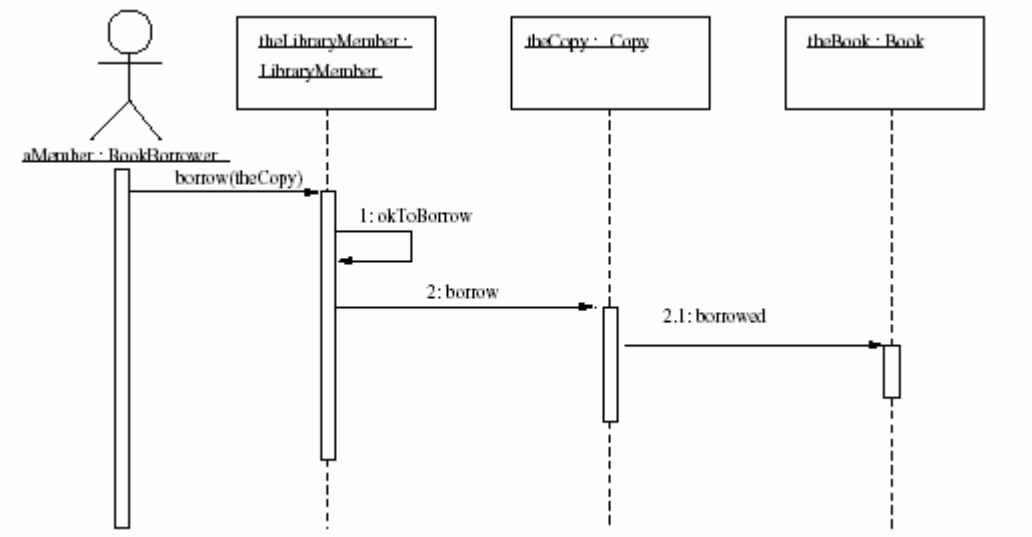
34. **Warum soll man gerade UML nehmen?** [defacto Standard in vielen Firmen| Unterstütz durch diverse Tools z.b. von der Firma Rational| Es ist mehr oder weniger intuitiv]
35. **Was sind die Schwächen von UML?** [Keine Semantik, was bedeutet das Modell? | Schlechte Unterstützung für Analyse| Unausgegorene und Cryptische Details die noch aus der Entwicklungsphase stammen]
36. **Was versteht man unter Use Cases?** [Stellt die Anforderungen und den entsprechenden Benutzer dar. Benutzer ist etwas ausserhalb des Systems. Z.B. Mensch oder andere Systeme| Entwickelt von Jacobson (1990) basiert auf der Idee von Szenarien| Actors sind

Prototypische Benutzer in bestimmten Rollen| Use Cases sind die Aufgaben der Actors



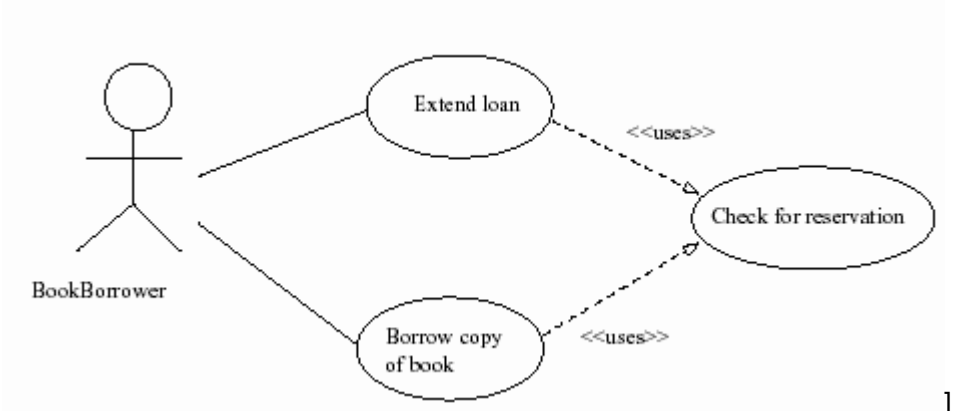
]

a. **Wo werden Use Cases eingesetzt?** [Bei der Anforderungsanalyse, allerdings eignet sich das nur für einen Teil]
b. **Welche Ergänzungen gibt es zu Use Cases?** [Text oder andere Modell zum Beispiel Sequence Diagram:



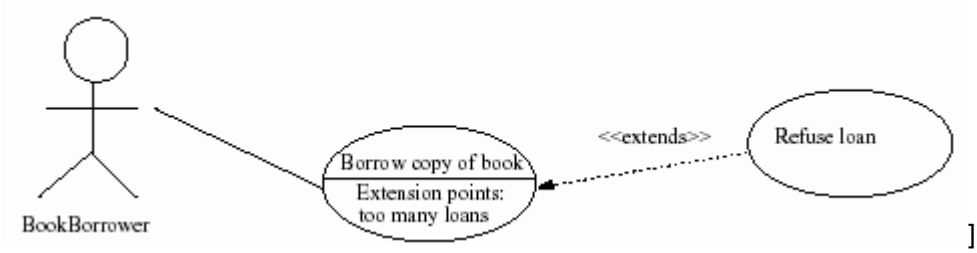
Denkbar wären auch zusätzlich unterschiedliche Symbole für Unterschiedle Actors, Schnittstellen und Prozesse]
c. **Wie erkennt man wichtige Use Cases?** [Es sollten immer bestimmte Dienste von gewissem Wert sein um das System nicht zu trivial zu halten]
d. **Viele Abläufe enthalten immer wieder die gleichen Komponenten, muss man die jedes Mal mit reinschreiben oder kann man bereits vorhandene wieder verwenden?** [<<uses>>: Man kann häufige Teile wieder verwenden, allerdings muss man diese ausfaktorisieren, damit man auch mit anderen Use Cases darauf zugreifen kann. Bei der Buchausleihe sowie bei der Verlängerung wird z.B. jedes Mal getestet

ob das gewünschte Buch nicht evtl. schon reserviert ist.



]

e. **Oft gibt es einen Use Case in verschiedenen Varianten, wie kann man das darstellen?** [<<extends>>: z.B. Use Case „Bezahlen“ <<extends>> z.B. auf Bar, Scheck, Überweisung, Kreditkarte etc.

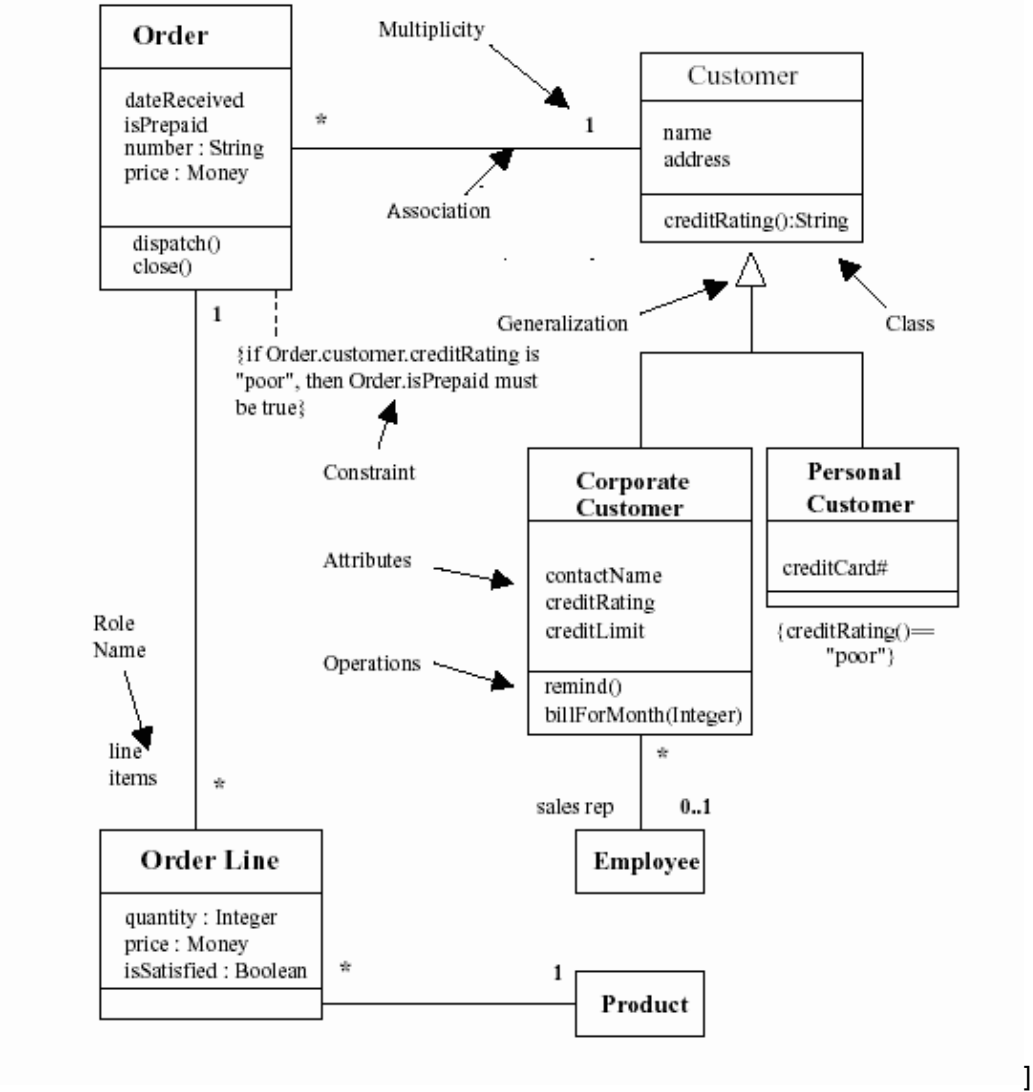


]

f. **Was macht man, wenn man z.B. bei der Buchausleihe nicht mehr zwischen Mitarbeitern und Kunden unterscheiden will?** [Generalizes: d.h. z.b. Ein Mitarbeiter wird zum normalen Ausleiher generalisiert]

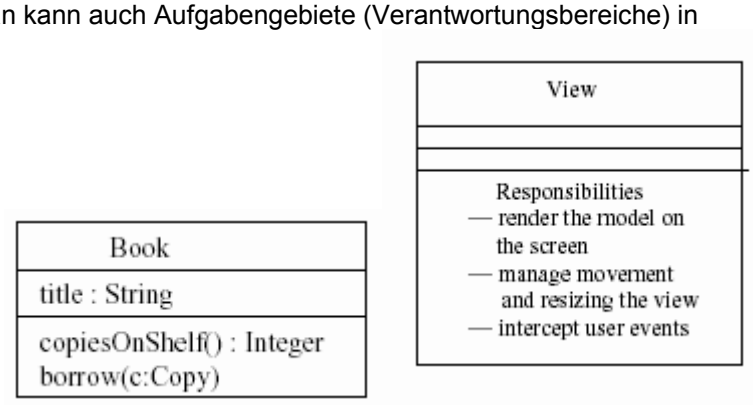
37. **Aus was besteht ein Objekt?** [Es hat einen Zustand, ein bestimmtes Verhalten und eine Identität, d.h. Namen, es existiert]
38. **Was ist die Schnittstelle eines Objekts?** [Die Schnittstelle beschreibt die Art der Nachrichten die ein Objekt erhalten kann, man unterscheidet zwischen private und public Interface. Die privaten Interfaces können nur vom Objekt selber zugegriffen werden]
39. **Was ist ein „Class Diagram“?** [Das Klassendiagramm beschreibt die Art der Objekt in einem System und deren unterschiedliche statische Beziehungen. Es geht also um die

statische Sich eines System. Wichtigste Beziehungen sind Associations und Subtypes.



]

40. **Wozu werden Klassendiagramme verwendet?** [Konzeptuell: Stellt Konzepte im Bereich dar. Keine direkte Beziehung zu einer Implementierung| Spezifikation: Spezifiziert Schnittstellen und Teile der Semantik. |Implementierung. Beschreibt was implementiert wird.]
41. **Was sind Klassen?** [Eine Klasse beschreibt eine Menge von Objekten mit äquivalenten Rollen in einem System.]
42. **Wie wir so eine Klasse visuell dargestellt?** [Als Viereck mit oder ohne Attribute (Daten bzw. Zustand des Objektes) bzw. Operationen/Methoden (definieren wie Objekte aufeinander wirken): Man kann auch Aufgabengebiete (Verantwortungsbereiche) in



texueller Form angeben:]
43. **Wann ist ein Klassenmodell gut?** [Die Objekte sollen die gewünschten Anforderungen erfüllen| Um Wartbarkeit und wieder Verwendbarkeit zu erhöhen, sollen Klassen die dauerhaften Objekte in dem Bereich darstellen.]

a. **Wie baut man ein gutes Klassenmodell auf?** [Göttliche Inspiration ☺| Data Driven Design: Identifizieren Sie alle Systemdaten und teilen sie diese in Klassen ein, danach überlegen sie sich Operationen| Responsibility Driven Design➔ Man beginnt zuerst mit Operationen oder sogar mit Responsibilities| Man kann auch die schriftliche Anforderungsanalyse analysieren und sämtliche Substantive und Hauptsätze herausschreiben und probieren Klassen herauszuextrahieren, Redundante, unklare Information kann gestrichen werden – Des Weiteren kann alles gestrichen werden was Ereignisse, Operationen sind oder außerhalb des Systemumfangs ist.]

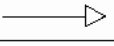
44. **Was sind CRC (Class – Responsibilities – Collaborators) Karten?** [Für jede mögliche Klasse notiert man auf einer Karte ihren Namen, ihren Verantwortungsbereich und Beziehungen – Wenn es zu viele Verantwortungen und Beziehungen gibt, muss man neue Klassen kreieren| Mit Hilfe der CRC Karten kann man im Entwicklungsteam ein Use Case Szenario aufbauen und Lücken erkennen – Erst ganz am Schluss werden Attribute und Methoden eingefügt

COPY	
Responsibilities	Collaborators
Maintain data about a particular book copy Inform corresponding BOOK when borrowed and returned	BOOK

]

45. **Welche 5 Beziehungstypen gibt es in UML zwischen Objekten bzw. Klassen?**

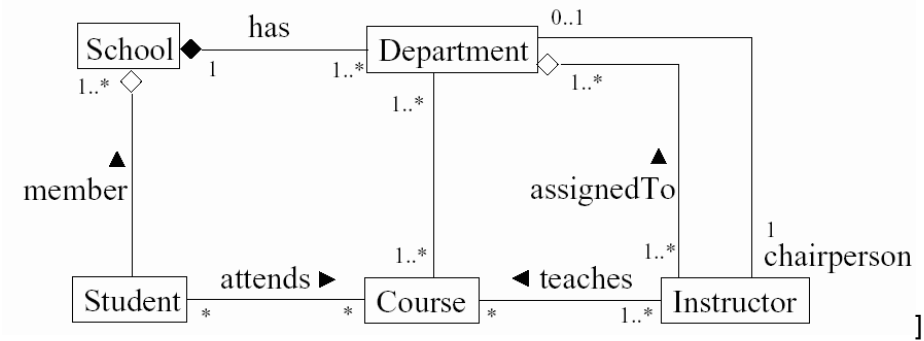
[

Relationship	*Function*	*Notation*
Association	Describes connection between instances of classes	
Generalization	A relationship between a more general description and a more specific variety	
Dependency	A relationship between two model elements	
Realization	Relationship between a specification and its implementation	
Usage	Situation where one element requires another for proper functioning	

]

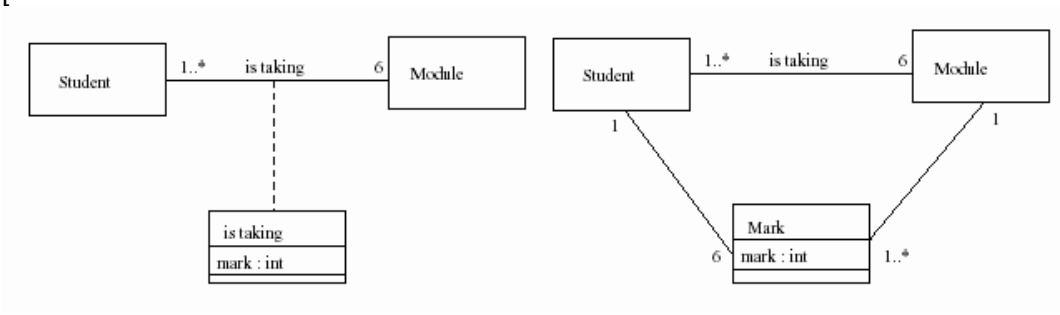
a. **Was sind Assoziationen?** [Beschreiben Verbindungen zwischen Objekten oder anderen Instanzen in einem System
b. **Wie kann man dieses assoziationen genauer beschreiben?** [Assoziationen können auch beschrieben werden also z.B. die Rolle die ein Objekt in einer einer Beziehung hat (e.g. Employee-Employer) Dies kann natürlich auch gerichtet sein, dann wenn die Beschreibung z.B. lautet „arbeitet für"]
c. **Was passiert wenn ein Objekt zu mehreren Instanzen eine Objektes Beziehungen unterhält?** [Ein Objekt kann eine Vielzahl von Assoziationen zu einem anderen Objekt haben e.g. Student belegt mehrere Kurse| Assoziationen können auch gerichtet sein, so ist es z.B. möglich dass ein Student eine Nachricht von seinem Kurs erhält, er aber keine Message zurückschreiben kann]
d. **Was ist der Unterschied zwischen Aggregation und Composition?** [Aggregation ist ein Anhäufung mehrerer Teile zu einem gesamt, man sagt als das Teil „Ist Teil des" Gesamten.Das Teil kann aber ohne das Gesamte weiterexistieren z.B. Student und Universität (Studenten können sehr gut ohne die Uni leben – Bei der Composition geht das nicht, der Teil kann nicht ohne das Gesamte überleben z.b. Institute können nicht ohne die zugehörig Universität

existieren



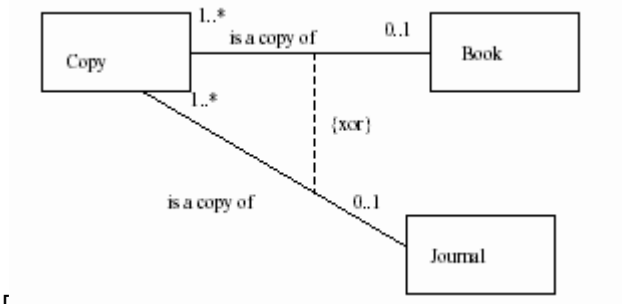
]

e. **Wie kann man Eigenschaften einer Assoziation modellieren?**
[



]

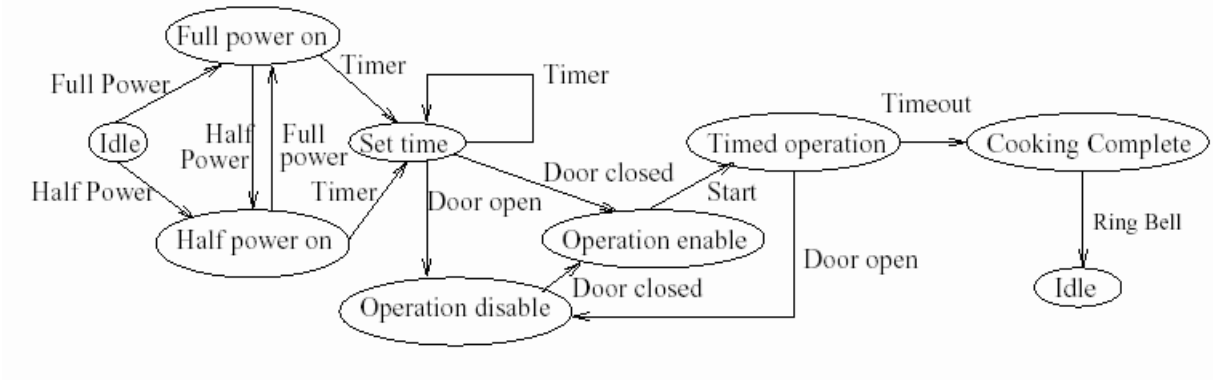
f. **Wie kann man Constraints darstellen um die Semantik genauer festzlegen?**



[Es gibt noch eine Erweiterung namens OCL bei dem Constraints durch logische Formeln spezifiziert werden (Verwenden wir aber nicht, wir verwenden Z)]

g. **Wie kann man Generalisierung einer Sub- zu einer Superklasse ausdrücken?** [is-a Beziehung]

46. **Was ist der Unterschied zwischen dynamischer und statischer Modellierung?** [Kontrolle und zeitliche Koordinierung. Es werden nun Zustände, Ereignisse, Zustandsübergänge und Aktivitäten modelliert]
47. **Wie kann man das Darstellen?** [Automaten spezifiziert den Kontrollfluss in Programmen: Zustände = Systemfunktionen, Übergänge = Ereignisse ➔ Die Zustände und Ereignisse müssen separat Erläutert werden.

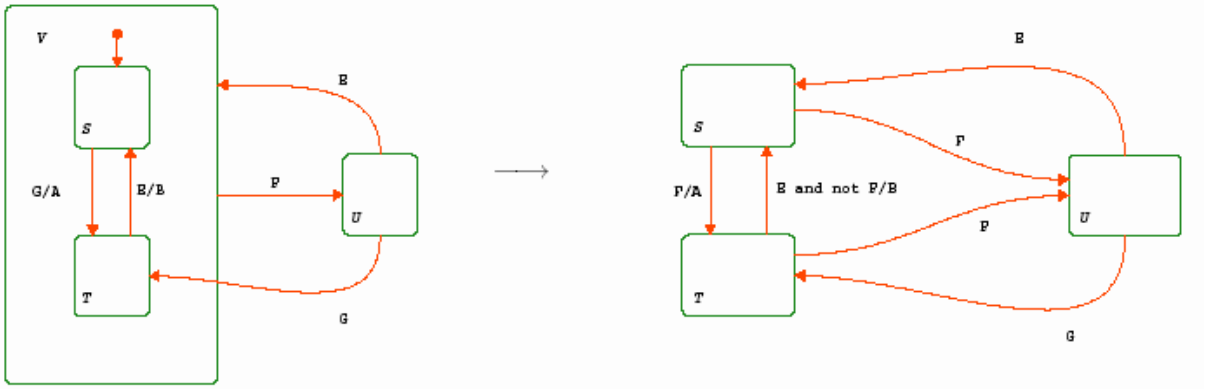


]

48. **Welchen Vorteil haben State Charts gegenüber State Machines (Automaten)?** [Hierarchische Anordnung möglich, die Entwicklung durch iterative Verfeinerung unterstützt| Parallele Vorgänge möglich darzustellen]
49. **Was können State Charts noch?** [Sie können erklären wie und wann ein Objekt einer Klasse reagiert| Sowohl in den Übergängen als auch in den Zuständen können Aktionen spezifiziert werden. Aktionen wiederum können durch Randbedingungen beschränkt werden (z.B. notlastcopy)]
50. **Was bedeuten die einzelnen Elemente eines Statecharts?**

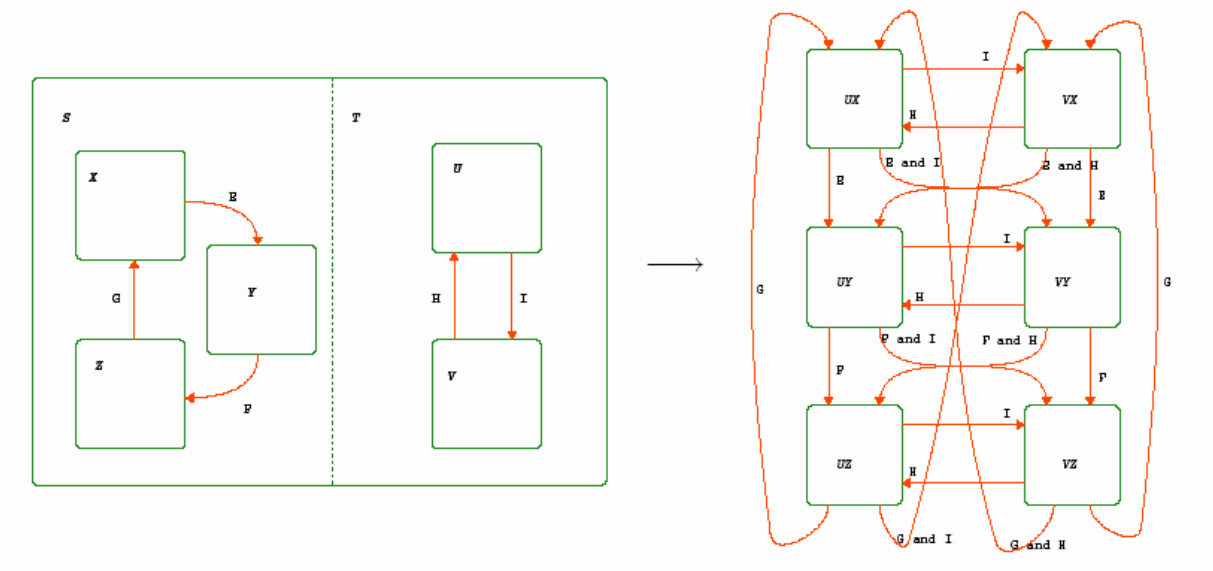
[Übergang zu S_2 findet statt, wenn das System im Zustand S_1 ist, Ereignis E gerade auftritt und die Bedingung C erfüllt ist.]

51. **Wie wird Hierarchie dargestellt?** [Man kann in mehreren Zuständen gleichzeitig sein, nämlich z.B. auch im Parent-Zustand:



]

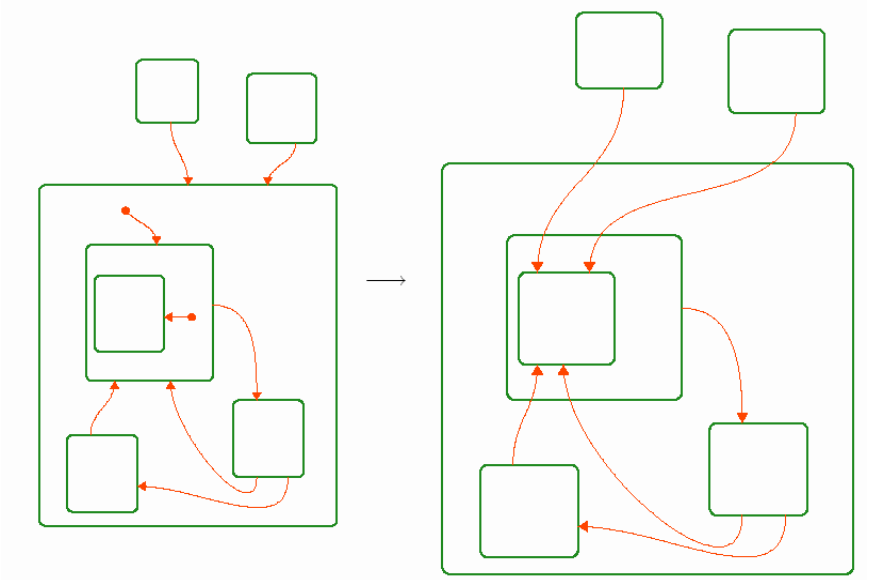
52. **Wie werden paralle Vorgänge dargestellt?** [UND-Zustände: Gleichzeitig in allen Zuständen des aktuellen Levels (hier gleichzeitig in S und T



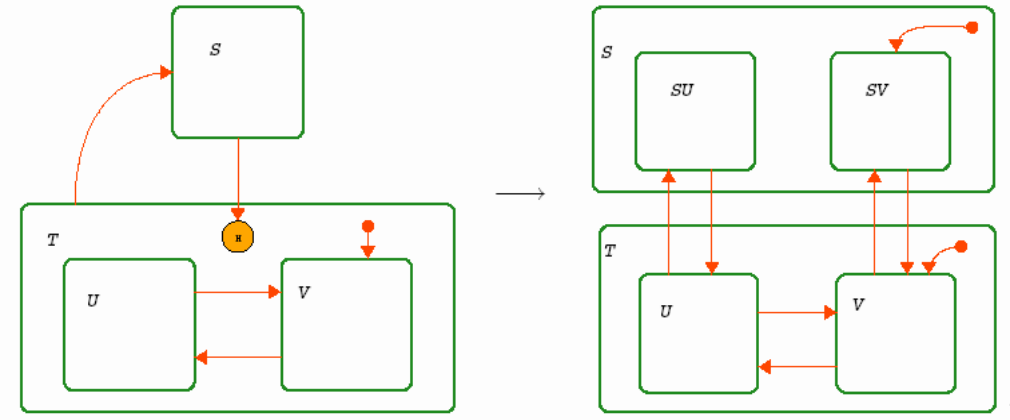
]

53. **Kann es Probleme bei der parallelen Ausführung geben?** [z.B. Wenn in einem Zustand ein Wert erhöht wird und im anderen gleichzeitig erniedrigt]

54. **Was macht der Default Connector?** [Er bestimmt wo es beim Sprung in ein Subsystem weitergeht

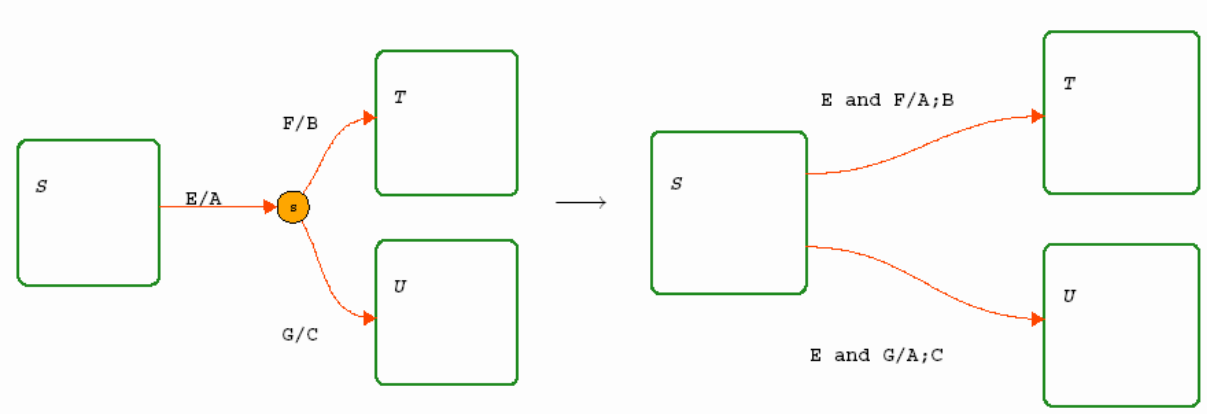


55. **Was macht der History Connector?** [Der History Connector merkt sich beim Verlassen des Subsystems welchen Zustand er als letztes hatte, damit bei erneutem Betreten des Subsystems dieser Zustand wiederhergestellt werden kann.



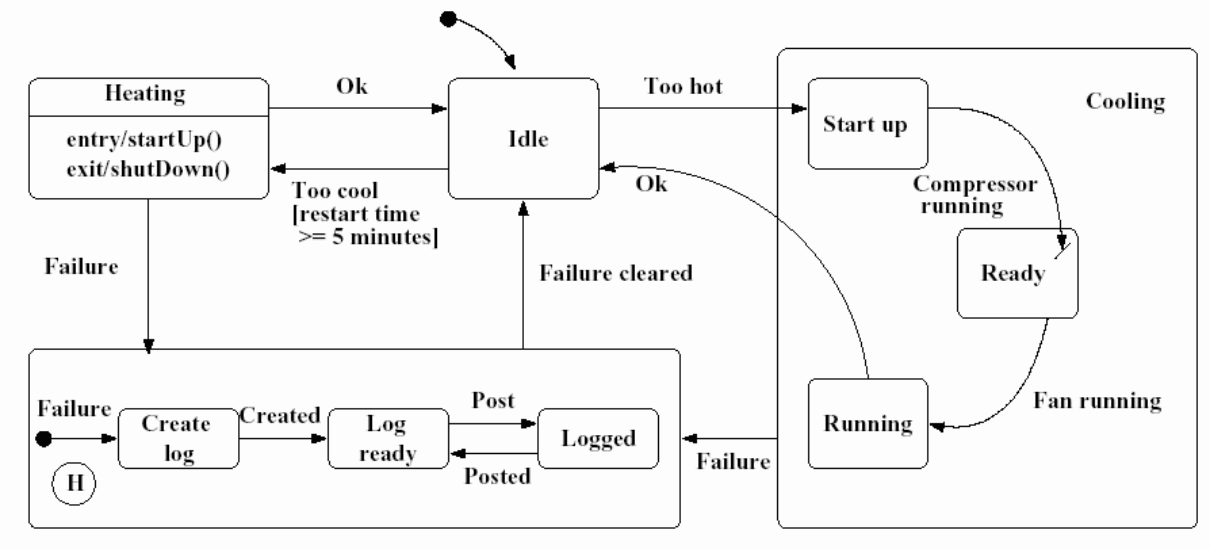
]

56. **Welche Aufgabe hat der Switch Connector?** [Wenn zwei Ereignisse darüber entscheiden, in welchen Zustand nach diesen Ereignissen gesprungen wird, dann setzt man einen Switch-Connector, es werden dann beide Aktionen ausgeführt



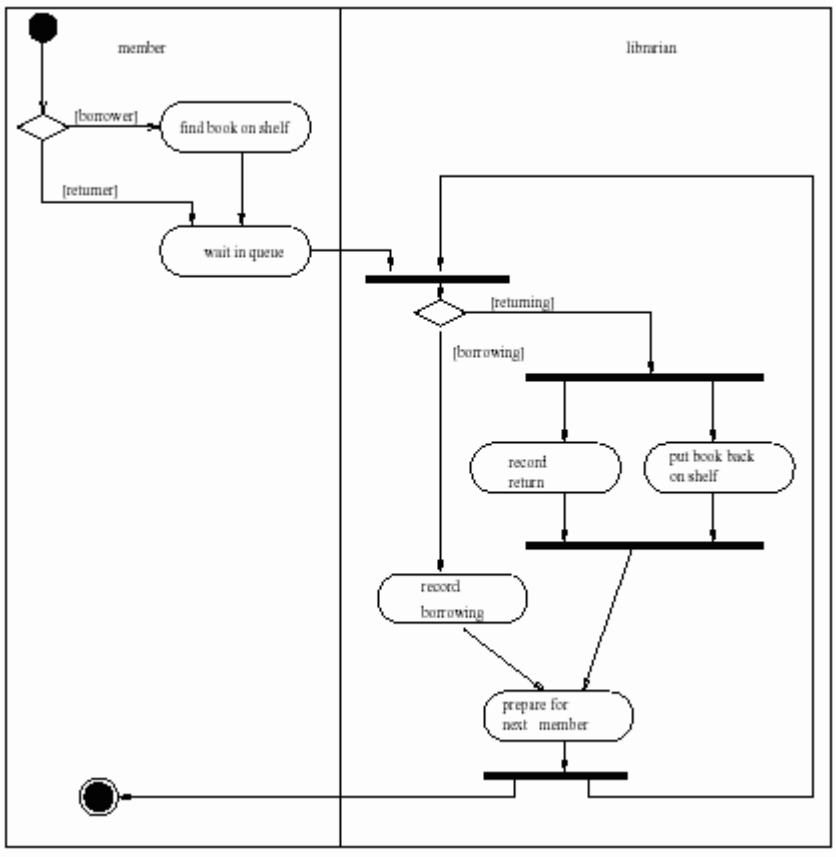
]

57. **Nennen Sie ein Beispiel für ein komplettes State-Chart Diagramm!** [Klimaanlage mit Fehlererfassung:



]

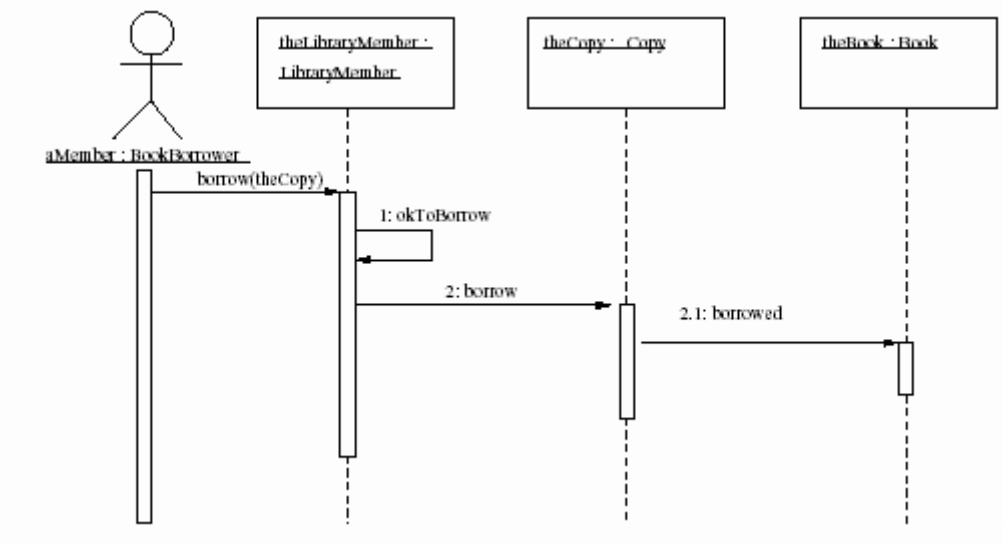
58. **Was sind Activity Diagrams und was ist der Unterschied zu Statecharts?** [Sehr ähnlich wie die Statecharts, sie lassen sich sogar in diese umwandeln. Unterschied: Die Synchronisation innerhalb und zwischen den Objekten wird betont (Synchronization Bars| „Entscheidungsraute“ d.h. Verzweigung, bei der in Abhängigkeit einer Bedingung in den einen oder den anderen Zustand gesprungen wird, Alternative zu Guards bei Statecharts| Swimmlanes beschreiben welches Objekt welche Aktivitäten ausführt, z.B. bei der Bücherei ist ein Swimlane der Borrower und die andere Swimlane der



librarian|]

59. **UML stellt zwei zwei verschiedene Modelle zur Verfügung um Interaktionen zwischen Modellen zu modellieren. Sequenz Diagramme und Interaktionsdiagramm. Die Sequenzdiagramme wurden genauer besprochen, wie funktionieren diese?** [Zeigt wie Objekte durch Nachrichtenaustausch während der Ausführungsszenarien interagieren. Beschreibt auch Objekt-Zeit Verhalten und (optional) die Lebenszeit in Szenarien. Sehr gut geeignet für CRC Karten. Die Syntax besteht aus Namen von Objekte, die Lifeline des Objektes mit dessen Aktivierungsrechteck und Pfeile die Nachrichtenaustausch anzeigen.

Weitere Kommentare sind möglich



]

60. **Welche Beschränkungen gibt es bei Sequence Diagrams?** [Verzweigungen mit Constraints sind möglich aber schwierig, d.h am Besten nur ein Szenario| Keine Hierarchien möglich| Man kann nur beschreiben wass passieren kann aber nicht was passieren soll]
61. **Welche Erweiterungen gibt es noch bei Sequence Diagrams?** [Erstellen und Löschen von Objekten| Randbedingungen, z.B. das Ausleihen eines Buches darf nicht länger als 5 Sekunden gehen.]
62. **Was ist eine formale Sprache?** [Eine Sprache wird formal genannt, wenn für sie eine formale Sprache (Syntax) festgelegt ist, deren Bedeutung (Semantik) mathematisch genau beschrieben ist]
63. **Warum Formale Sprachen und nicht die intuitiveren Semi-Formalen Sprachen?** [Formale Sprache entfernen Mehrdeutigkeiten, fügen Präzision hinzu, strukturieren Information auf geeigneter Abstraktionsebene, sie unterstützen das Beweisen von Designeigenschaften und werden technisch direkt von verschiedenen Tools und Systemen unterstützt – Obwohl Formale Sprachen durch ihren mathematischen Charakter teuer und kompliziert wirken, lohnt sich die Mühe]
64. **Was kann man mit formalen Sprachen machen?** [Sie unterstützen das Design, die Entwicklung, die Verifikation, das Testen und die Wartung eines Systems]
65. **Welche Formalen Sprachen kennen Sie?** [Z, PL1=, Gleichungslogik, Hornlogik, HOL
66. **Was ist Z?** [Z ist eine sehr ausdrucksstarke formale Sprache, sie basiert auf der „Prädikatenlogik erster Stufe mit Gleichheit (PL1=)“ und getypter Mengentheorie. Z stellt diverse mathematische Werkzeuge (Bibliothek von mathematischen Standarddefinitionen und abstrakten Datentypen) zur Verfügung. Z unterstützt strukturiertes modellieren eines Systems, sowohl statisch als auch dynamisch. Verfeinerungsprozess kann bis hin zu einem ausführbaren Code weitergeführt werden. Die Z-Spezifikation identifiziert legale und illegale Daten und Operationen. Z wird von vielen Tools und Systemen unterstützt – Seit 1989 eingeführt, ist bald ISO Standard]
67. **Was kann Z nicht?** [Z ist nicht geeignet für die Beschreibung von nich-funktionalen Eigenschaften wie Benutzebarkeit, Effizienz, Größe, Zuverlässigkeit usw. Und zur Beschreibung von zeitlichem oder nebenläufigem Verhalten. Dafür sind andere Methoden die man mit Z kombinieren kann viel besser geeignet]
68. **Was drückt Z aus?** [Der Schlüssel ist die Abstraktion, das Verstehen des Codes ist von dem Verstehen seiner Funktion zu trennen. Genau dieses Verstehen der Funktion drückt Z aus. Beispiel.: Stellen sie sich einen Videorekorder vor, dessen einzige Dokumentation eine Blaupause seiner Elektronik ist ➔ Falsche Abstraktionsebene zum Verstehen. Man kann mit Z spezifizieren was das System tun muss ohne festzulegen wie]
69. **Wie kommt man in 3 Schritten zur Z-Spezifikationen?** [Definiere Hilfsfuntkionen und -typen des Systems| Definieren den Zustandsraum des Systems| Definiere die Operationen des Systems (Basierend auf den Relationen des Zustandsraumes]
70. **Was bedeuten die folgenden Darstellungen?**
    [

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$
$birthday' =$
$\quad birthday \cup \{name? \mapsto date?\}$

Name of operation (schema)

structured import (symbol Δ)
input of operation (symbol ?)
input of operation (symbol ?)

---

precondition for success of operation
extend the birthday function
(if precondition is satisfied)

FindBirthday

$\Xi BirthdayBook$
$name? : NAME$
$date! : DATE$

$name? \in known$
$date! = birthday(name?)$

Name of operation (schema)

structured import (symbol Ξ)
input of operation (symbol ?)
output of operation (symbol !)

---

precondition for success of operation
output of operation (if successfull)

FindBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date! : DATE$

$known' = known$
$birthday' = birthday$
$name? \in known$
$date! = birthday(name?)$

Remind

$\Xi BirthdayBook$
$today? : DATE$
$cards! : \mathbb{P}\, NAME$

$cards! = \{ n \in known \mid birthday(n) = today? \}$

InitBirthdayBook

$BirthdayBook$

$known = \varnothing$

]

71. **Welches sind die wichtigsten Ausdrücke (E) und Prädikate (P) von Z?**

[

$$
\begin{array}{lcll}
E & ::= & c_x \mid v_x & [\text{constants, variables}] \\
& \mid & E\,E & [\text{application}] \\
& \mid & (E, \ldots, E) & [\text{cartesian product}] \\
& \mid & \{v_1 : E;\ \ldots;\ v_n : E \mid P \bullet E\} & [\text{comprehension}] \\
& \mid & (\!| \ tag_1 \rightsquigarrow E, \ldots, tag_n \rightsquigarrow E \ |\!) & [\text{tagged records (bindings)}] \\
& \mid & E.tag & [\text{element selection in record}]
\end{array}
$$

$$
\begin{array}{lcll}
P & ::= & true \mid false \mid c_x(E, \ldots, E) & \\
& \mid & \neg\, P \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid P \Leftrightarrow P & \\
& \mid & \forall x : S \bullet P \mid \exists x : S \bullet P & [\forall \text{ declaration} \bullet \text{predicate}]
\end{array}
$$

]

72. **Wie definiert man eine Menge bei der getypten Mengentheorie?** [Zumindest kleine Menge kann man recht einfach definieren per Extension z.B. Oceans == [atlantic, arctic, indian, pacific] ➔ n ist der (neue) Name für den Ausdruck e ➔ n == e]
    a. **Wann sind zwei Mengen in getypter Mengentheorie gleich?** [Genau dann wenn sie die gleichen Elemente enthalten bzw.

    $$S = T \quad \text{iff} \quad (\forall x : S \bullet x \in T) \text{ and } (\forall x : T \bullet x \in S)$$

    x darf nicht frei in S oder T vorkommen]
    b. **Wann gehört ein Ausdruck E zu einer per Extension definierten Menge?** [gdw. er einem der Elemente der Menge gleicht bzw.

    $$e = u_1 \vee \ldots \vee e = u_n \quad \text{iff} \quad e \in \{u_1, \ldots, u_n\}$$

    Reihenfolge und Häufigkeit der Elemente ist unwichtig]
    c. **Was bedeutet folgender Ausdruck?** $\{x : S;\ y : T \mid p \bullet e\}$ [Die Variablendeklaration links vor dem Strich ist die Grundmenge, sie wird über das Prädikat (auch Filter genannt) ausgesiebt. Die entstehende Menge wird dann über einen Ausdruck e kombiniert, sodass eine neue Menge entsteht. Z.B. ergibt sich aus

    $\{i : \mathbb{N} \mid (i > 4) \bullet (2 * i) + 1\}$ die Menge [11,13,15,17,…]
    d. **Wie ist die Potenzmenge definiert?** [Die Potenzmenge von S ist die Menge aller Teilmengen von S inkl. Der leeren Menge: z.B.:

    $$\mathbb{P}\{x, y\} = \{\varnothing, \{x\}, \{y\}, \{x, y\}\}$$

    ]
        i. **Welches Axiom lässt sich dadurch ableiten?**

        [$\text{Axiom:} \quad T \subseteq S \quad \text{iff} \quad T \in \mathbb{P}\, S$]
    e. **Was ist das Kartesische Produkt S × T?** [S x t besteht aus allen Tuppeln (x,y), wobei x ∈ S und y ∈ T diese kann man auch so schreiben:

    $$S \times T = \{x : S;\ y : T \mid true\}$$

    ]

i. **Welche Axiome leiten sich daraus ab?**
[
- $(x_1, \ldots, x_n) \in (S_1 \times \ldots \times S_n)$ iff $(x_1 \in S_1) \wedge \ldots \wedge (x_n \in S_n)$
- $t = (x_1, \ldots, x_n)$ iff $(t.1 = x_1) \wedge \ldots \wedge (t.n = x_n)$ (projection)

]

ii. **Nennen Sie die Axiome für Union, Intersection und Difference!**
- $x \in (S \cup T)$ iff $(x \in S) \vee (x \in T)$
- $x \in (S \cap T)$ iff $(x \in S) \wedge (x \in T)$
- $x \in (S \setminus T)$ iff $(x \in S) \wedge (x \notin T)$
- etc.

[ ]

73. **Welche Typen gibt es in Z?** [Eigentlich nur einen, nämlich die Menge der ganzen Zahlen $\mathbb{Z}$, alle anderen Typen werden aus $\mathbb{Z}$, den Basistypen und den freien Typen gebildet:

$$
\begin{array}{lcll}
\tau_B & ::= & \mathbb{Z} & [\text{the integers}] \\
& | & B & [\text{Basis types and free types}] \\
& | & \mathbb{P}\,\tau_B & [\text{Power}-\text{set types}] \\
& | & \tau_B \times \ldots \times \tau_B & [\text{Product types}] \\
& | & [tag_1 \rightsquigarrow \tau_B, \ldots, tag_n \rightsquigarrow \tau_B] & [\text{Tagged record types}]
\end{array}
$$

]

a. **Was sind Basistypen?** [Interne Struktur ist unsichtbar, z.B. [NAME]]
b. **Was sind Freie Typen?** [eine Aufzählung von Konstanten, z.B.: COLORS::= red|orange|yellow|green|blue|indigo|violet]
c. **Was sind Produkttypen?** [T x U ist der Typ aller (geordneten) Paare aus Elementen von T und Elementen von U. z.B.
   * by definition: $\_ + \_$ has type $(\mathbb{Z} \times \mathbb{Z}) \rightarrow \mathbb{Z}$, and $(3, 4)$ has type $\mathbb{Z} \times \mathbb{Z}$.
   * $COLORCODE == COLORS \times \mathbb{Z}$ has type $\mathbb{P}(COLORS \times \mathbb{Z})$.

   ]

74. **Was muss gegeben sein, damit Type-Checking Algorithmen möglich sind?** [Type Checking Algorithmen sind möglich, weil jeder Wert X in einer Spezifikation genau einem Typ zugeordnet wird. Z.B. {2,4,red,yellow,6} würde einen Type Error geben, da Typen vermischt sind, d.h. alle Elemente einer Menge müssen den selben Typ haben!]
75. **Für was werden Urteile (Judgements) und Inferenzregeln benötigt?** [Sie ermöglichen wohlgetypte/-formierte Ausdrücke und Prädikate zu identifizieren. Z.B. kann man folgendes damit ausschließen: $3 < \varnothing$ and $5 \wedge \mathbb{Z}$. und man kann z.B. Beweisen dass: $\_ + \_(3, 4) :: \mathbb{Z}$,]
76. **Wie definiert man Abkürzungen in Z und wozu sind sie?** [Ein Abkürzung (Symbol) stellt einen Term dar, d.h. alles, den Term erfüllende steht dann im Symbol z.B. SEInstructors == {Basin, Vigano, Wolff}| Ein Symbol kann auch mit einem (Eingabe) Parameter gepaart werden z.B. eine Menge, mit der irgendetwas gemacht wird]
77. **Nennen Sie ein Beispiel für eine Axiomatische Definition in Z, welche die natürlichen spezifiziert!** [Aus der Grundstruktur

$$\frac{x : S}{p} \qquad \frac{\text{Declaration}}{\text{Predicate (axiom for object } x : S)}$$

$$\frac{\mathbb{N} : \mathbb{P}\,\mathbb{Z}}{\forall z : \mathbb{Z} \bullet (z \in \mathbb{N}) \Leftrightarrow (z \geq 0)}$$

ergibt sich dann]

78. **Was versteht man unter einer Generischen Axiomatischen Definition?** [Zusätzlich kommt noch ein (Eingabe-)Parameter hinzu. Es kann verschiedene Parameter,

$$\frac{[X]}{\begin{array}{|l} x : X \\ \hline p \end{array}}$$

Deklarationen und Prädikate geben, Geschrieben wird das so:

$$\frac{\text{Declarations}}{\begin{array}{|l} \text{Predicate} \\ \ldots \\ \text{Predicate} \end{array}} \qquad \frac{d1, d2, d3 : \mathbb{Z}}{\begin{array}{|l} d1 + d2 = 7 \\ d1 < d2 \\ d1 + d3 = 0 \end{array}}$$

z.B.
dieses Beispiel entspricht der logischen Formel:

$$(d1 : \mathbb{Z}) \wedge (d2 : \mathbb{Z}) \wedge (d3 : \mathbb{Z}) \wedge (d1 + d2 = 7) \wedge (d1 < d2) \wedge (d1 + d3 = 0)$$

]

79. **Was genau ist ein Objekt in Z?** [Alle Objekte in Z sind Mengen ➔ Prädikate werden durch die Mengen von Objekten die sie erfüllen definiert

$$\frac{crowds : \mathbb{P}(\mathbb{P}\, Person)}{crowds = \{s : \mathbb{P}\, Person \mid \#s > 2\}} \qquad (\ \# \text{ is } size \text{ or } cardinality \text{ of set } s)$$

so that

$$\{Jack, Janet, Chrissy\} \in crowds \quad \text{is } true$$
$$\{Adam, Eve\} \in crowds \quad \text{is } false$$

]

80. **Gibt es in Z einen Datentyp Boolean? Warum?** [Nein, wird auch nicht benötigt, denn es kann zu Verständnisproblemen führen stattdessen verwendet man deskriptive binäre Aufzählungen, sprich die Möglichkeiten. Es macht also mehr zu fragen, ob die Tür einer Mikrowelle offen bzw. zu ist, anstelle Tür=True bzw. False. Wir können also z.B. schreiben:
$(power = on) \Rightarrow (door = closed)$ |Anstatt eine boolesche Funktion zu definieren die die Eigenschaft testet, erstellt man eine Menge die die Eigenschaft erfüllt und testet ob das Element in der Menge ist➔ Beispiel mit Test auf ungerade Zahl]
81. **Was ist eine Relation und was versteht man unter dom und range?** [Eine Relation ist eine Abbildung zwischen zwei Mengen. Unter dom versteht man den Definitionsbereich der Abbildung. Unter Range versteht mant den Wertebereich einer Abbildung]
82. **Was versteht man bei einem Query unter Domain-/Range Restriction bzw. anti-restriction?** [Gibt an, welche Elemente der Domain/Range verwendet werden sollen (Restriction) oder welche ausgeschlossen werden sollen (Anti-Restriction)

Domain restriction $\_ \lhd \_$ and anti-restriction $\_ \lhd\!\!\!\!- \_$

range restriction $\_ \rhd \_$ and anti-restriction $\_ \rhd\!\!\!\!- \_$

Beispiel:

- $\{Basin, Wolff\} \lhd telephones = \{Basin \mapsto 8240, Basin \mapsto 8241, Wolff \mapsto 8247\}$
- $\{Basin, Wolff, Vigano`\} \lhd telephones \rhd \{8240, 8247\} = \{Basin \mapsto 8241, Vigano` \mapsto 8243\}$

]

83. **Wie überschreibt man bereits definierte Relationen?** [Mit Overriding:

$$[X, Y]$$
$$\_ \oplus \_ : (X \leftrightarrow Y) \times (X \leftrightarrow Y) \to (X \leftrightarrow Y)$$
$$\forall R, S : (X \leftrightarrow Y) \bullet$$
$$R \oplus S = ((\mathrm{dom}\, S) \lhd R) \cup S$$

z.B.:

$$telephones = \{Basin \mapsto 8240, \ldots, Brucker \mapsto 8247, Ayari \mapsto 8244, \ldots\} \Rightarrow telephones \oplus \{Brucker \mapsto 8248, Ayari \mapsto 8249\} = \{Basin \mapsto 8240, \ldots, Brucker \mapsto 8248, Ayari \mapsto 8249, \ldots\}$$

]

84. **Was muss man tun um nur das Bild einer Abbildung zu erhalten?** [Man will also die Ergebnismenge (das Abbild) unter Angabe der Wertemenge erhalten:

$$[X, Y]$$
$$\_(\!|\_|\!) : (X \leftrightarrow Y) \times (\mathbb{P}\, X) \to (\mathbb{P}\, Y)$$
$$\forall R : (X \leftrightarrow Y);\ A : \mathbb{P}\, X \bullet$$
$$R(\!|A|\!) = \{y : Y \mid \exists x : A \bullet x \mapsto y \in R$$

$$\Rightarrow \quad R(\!|A|\!) = \mathrm{ran}(A \lhd R)$$

Example:

$$telephones(\!|\{Basin, Wolff\}|\!) = \mathrm{ran}(\{Basin, Wolff\} \lhd telephones) = \{8240, 8241, 8247\}$$

]

85. **Was macht die Inverse auf Relationen?** [Vertauscht Source und Target einer Relation:

$$[X, Y]$$
$$\_^{\sim} : (X \leftrightarrow Y) \to (Y \leftrightarrow X)$$
$$\forall R : (X \leftrightarrow Y) \bullet$$
$$R^{\sim} = \{x : X;\ y : Y \mid (x \mapsto y \in R) \bullet y \mapsto x\}$$

- Example:
  $\{Wolff \mapsto 8247, Vigano' \mapsto 8243\}^{\sim} = \{8247 \mapsto Wolff, 8243 \mapsto Vigano'\}$

]

86. **Wie kann man zwei Relationen kombinieren?** [Wenn man eine Relation von A auf B und eine von B auf C, dann will man dies zu einer Relation kombinieren die A auf C abbildet:

$$[X, Y, Z]$$
$$\_ \fatsemi \_ : (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \to (X \leftrightarrow Z)$$
$$\forall R : (X \leftrightarrow Y);\ S : (Y \leftrightarrow Z) \bullet$$
$$R \fatsemi S = \{x : X, z : Z \mid (\exists y : Y \bullet (x \mapsto y \in R) \land (y \mapsto z \in S)) \bullet x \mapsto z\}$$

Example:
$telephones : Person \leftrightarrow Phone$ and $departments : Phone \leftrightarrow Department$
$\Rightarrow \quad \{Vigano' \mapsto software - engineering\} \in telephones \fatsemi departments$

]

87. **Ist die Komposition kommutativ?** [Nein, deshalb gibt es eine andere Regel:

$$[X, Y, Z]$$
$$\_ \circ \_ : (Y \leftrightarrow Z) \times (X \leftrightarrow Y) \to (X \leftrightarrow Z)$$
$$\forall R : (X \leftrightarrow Y);\ S : (Y \leftrightarrow Z) \bullet$$
$$S \circ R = R \fatsemi S$$

Rückwärtsgerichtete Komposition]

88. **Eine Komposition kann auch eine Iteration enthalten, wie sieht hier die Regel aus?** [Die Komposition wird mehrmals hintereinander ausgeführt und ruft sich selber iterativ immer wieder auf.

$$[X]$$
$$\mathsf{iter} : \mathbb{Z} \to (X \leftrightarrow X) \to (X \leftrightarrow X)$$
$$\forall R : (X \leftrightarrow X) \bullet$$
$$\mathsf{iter}\, 0\, R = \mathsf{id}\, X \land$$
$$(\forall i : \mathbb{Z} \bullet \mathsf{iter}(i+1)\, R = R \fatsemi (\mathsf{iter}\, i\, R)) \land$$
$$(\forall i : \mathbb{Z} \bullet \mathsf{iter}(-i)\, R = R \fatsemi (\mathsf{iter}\, i\, (R^{\sim})))$$

]

89. **Wie ist die transitive Hülle und die reflexiv transitive Hülle definiert?**
[

$$[X]$$

$$\_^+, \_^* : (X \leftrightarrow X) \to (X \leftrightarrow X)$$

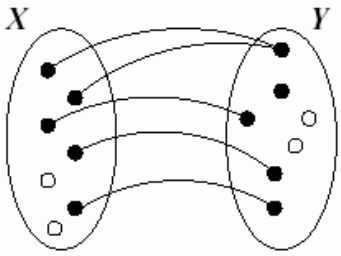
$$\forall R : (X \leftrightarrow X) \bullet$$
$$R^+ = \bigcap \{S : (X \leftrightarrow X) \mid (R \subseteq S) \land (S \fatsemi S \subseteq S)\} \land$$
$$R^* = \bigcap \{S : (X \leftrightarrow X) \mid (\mathrm{id}\, X \subseteq S) \land (R \subseteq S) \land (S \fatsemi S \subseteq S)\}$$

Transitive Hülle heisst, dass man bei einem Graphen, bei dem es beispielsweise eine Verbindung von a nach b und b nach c gibt indirekt auch von a nach c gibt. Die Transitive Hülle besteht also aus a➔b, b➔c und a➔c| Die reflexiv transitive Hülle enthält dann zusäzlich noch die Identitäten a➔a, b➔b und c➔c]

90. **Was sind denn Funktionen?** [Funktionen sind besondere Relationen, wo jedes Element einer Menge mit höchstens einem Element einer anderen Menge verbunden ist.]
91. **Was ist der Unterschied zwischen einer partiellen und einer totalen Funktion?** [Bei einer partiellen Funktion wird nur ein Teil der Definitionsmenge auf die Wertemenge abgebildet
wird

The set $X \pfun Y$ of the partial functions is:

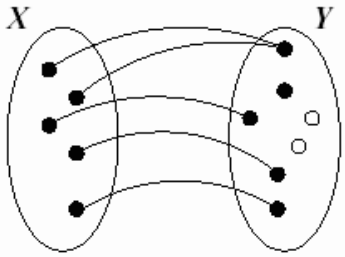
$$X \pfun Y ==$$
$$\{f : X \leftrightarrow Y \mid \forall x : X;\ y_1, y_2 : Y \bullet$$
$$(x \mapsto y_1 \in f) \land (x \mapsto y_2 \in f) \Rightarrow y_1 = y_2\}$$



| Bei einer einer totalen Funktion wird die komplette Definitionsmenge abgebildet

The set $X \to Y$ of all such functions is:

$$X \to Y == \{f : X \pfun Y \mid \mathrm{dom}\, f = X\}$$



]

92. **Nennen ein Beispiel für eine totale Funktion in Z-Notation!**

$$double : \mathbb{N} \leftrightarrow \mathbb{N}$$

$$\forall m, n : \mathbb{N} \bullet (m \mapsto n \in double) \Leftrightarrow (m + m = n)$$

[ ]

93. **Erklären sie die beiden Regeln der Funktionsapplikation app-intro und app-elim!**

$$\frac{\exists_1 p : f \bullet p.1 = a \quad (a \mapsto b) \in f}{b = f(a)} \text{ app-intro (if } b \text{ does not occur free in } a)$$

$$\frac{\exists_1 p : f \bullet p.1 = a \quad b = f(a)}{(a \mapsto b) \in f} \text{ app-elim (if } b \text{ does not occur free in } a)$$

[app-intro: Bedeutet dass wenn man eine Abbildung von a nach b hat, bei der die Funktionswerte in f liegen, dass diese Funktion f jedem a ein b zuordnet|app-elim ist genau umgekehrt, nämlich, dass wenn man eine Funktion f hat, die zu einem Definitionswert a einen Funktionswert b gibt, so gibt es eine Abbildung von a nach b mit Funktionswerten aus f]

94. **Was ist die Lamda Notation?** [Im Prinzip eine vereinfachung der Schreibweise, normalerweise gibt man an, was auf was abgebildet wird wie in folgendem Beispiel:

$f = \{x : X \mid p \bullet x \mapsto e\}$ | Dies kann vereinfacht in dem man x mit Lambda markiert und sozusagen sagt, dass x die abzubildende Variable ist. Die Schreibweise sieht dann so aus: $f = \lambda x : X \mid p \bullet e$ | In Z unterscheidet sich dann z.B. das „double“ Beispiel folgendermaßen:

$$\begin{array}{|l} double : \mathbb{N} \leftrightarrow \mathbb{N} \\ \hline double = \lambda m : \mathbb{N} \bullet m + m \end{array}$$

$$\begin{array}{|l} double : \mathbb{N} \leftrightarrow \mathbb{N} \\ \hline \forall m, n : \mathbb{N} \bullet \\ \quad (m \mapsto n \in double) \Leftrightarrow (m + m = n) \end{array}$$

]

95. **Welches sind die Eigenschaften der verschiedenen Funktionstypen!**

[

- Set $X \rightarrowtail Y$ of partial injections:

$$X \rightarrowtail Y == \{f : X \pfun Y \mid f^\sim \in Y \pfun X\}$$

- Set $X \rightarrowtail Y$ of total injections:

$$X \rightarrowtail Y == \{f : X \fun Y \mid f^\sim \in Y \pfun X\}$$

- Set $X \twoheadrightarrow Y$ of partial surjections:

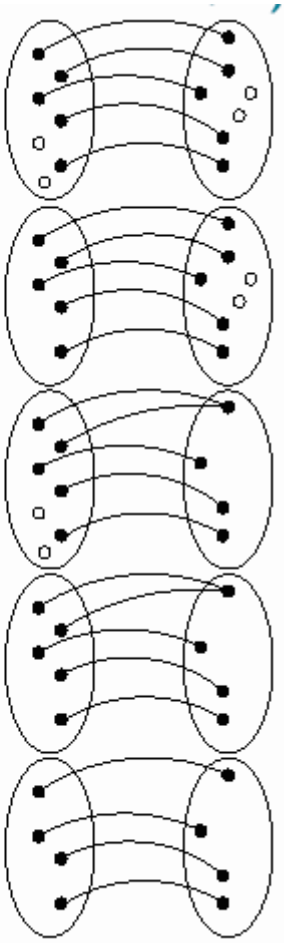
$$X \twoheadrightarrow Y == \{f : X \pfun Y \mid \mathrm{ran}\, f = Y\}$$

- Set $X \twoheadrightarrow Y$ of total surjections:

$$X \twoheadrightarrow Y == \{f : X \fun Y \mid \mathrm{ran}\, f = Y\}$$

- Set $X \rightarrowtail\!\!\!\!\!\twoheadrightarrow Y$ of total bijections:

$$X \rightarrowtail\!\!\!\!\!\twoheadrightarrow Y == (X \rightarrowtail Y) \cap (X \twoheadrightarrow Y)$$



]

96. **Wann nennt man eine Menge F endlich?** [Wenn die Menge bis zu einer bestimmten natürlichen Zahl n aufzählbar ist, d.h. wenn es eine totale Bijektion mit Definitionsmenge 1,2,…,n und Wertemenge F gibt]
97. **Wie drückt man in Z einen Zahlenbereich wie 1..10 aus?** [Mittels des „Number Range“

$$\_ \upto \_ : (\mathbb{N} \times \mathbb{N}) \fun \mathbb{P}\,\mathbb{N}$$

$$\forall\, m, n : \mathbb{N} \bullet m \upto n = \{i : \mathbb{N} \mid m \leq i \leq n\}$$

Operator (Upto) _..._:]
98. **Wie drückt man in Z die Menge aller endlichen Teilmengen aus?**
[$\mathbb{F}\, X == \{s : \mathbb{P}\, X \mid \exists\, n : \mathbb{N} \bullet \exists f : (1 \upto n) \twoheadrightarrow s \bullet true\}$ F ist dann die Menge aller endlichen Teilmengen von X| Wenn X bereits eine endliche Menge ist, dann ist $\mathbb{F}\, X = \mathbb{P}\, X$]
99. **Wie gibt man die Größe (Kardinalität) einer endlichen Menge an?** [mit dem Zeichen # :

$$[X]$$

$$\# : \mathbb{F}\, X \fun \mathbb{N}$$

$$\forall\, s : \mathbb{F}\, X;\ n : \mathbb{N} \bullet n = \# s \Leftrightarrow \exists f : (1 \upto n) \twoheadrightarrow s \bullet true$$

]
100. **Was ist eine endliche Funktion und wie drückt man diese Eigenschaft in Z aus?** [Eine endliche Funktion hat einen endlichen Definitionsbereich (Domain), diese Eigenschaft word

in Z folgendermaßen ausgedrückt:

$$A \twoheadrightarrow B == \{f : A \pfun B \mid \operatorname{dom} f \in \mathbb{F}\, A\} \quad \text{or} \quad A \rightarrowtail\!\!\!\!\!\twoheadrightarrow B == (A \twoheadrightarrow B) \cap (A \rightarrowtail B)$$

]

101. **Was ist eine Sequenz und wie definiert man diese in Z?** [Im Gegensatz zu einer Menge ist eine Sequenz eine geordnete Collection von Elementen, z.B. die Monate Januar bis Dezember.Zur Sequenz (also in Reihenfolge) werden die Monate durch den seq Operator in Z, z.B.:

$$year : \operatorname{seq} MONTHS$$

$$year = \langle\, jan, feb, mar, apr, jun, jul, aug, sep, oct, nov, dec \,\rangle$$

Der seq Operator definiert uns als eine Ordnung in einer Menge, weitere Beispiele sind z.B.:

- $\langle\langle feb, mar\rangle, \langle\rangle, \langle apr\rangle\rangle \in \operatorname{seq}(\operatorname{seq} MONTHS)$ [empty sequence $\langle\rangle$]
- $\langle 77, 5, 6, 18, 43, 61\rangle \in \operatorname{seq}\mathbb{N}$
- $\langle\{1, 2, 83\}, \varnothing\rangle \in \operatorname{seq}(\mathbb{P}\,\mathbb{N})$

102. **Wann ist eine Menge von Sequenzen endlich?** [Wie eben, d.h. es muss eine bijektive Abbildung von den natürlichen Zahlen auf die Elemente der Sequenz geben, wobei die natürlichen durch eine Zahl n beschränkt sein müssen]
103. **Wie kann man zwei Sequenzen in Z aneinander hängen?** [Mit der Concatenation:

$$[X]$$

$$\_ \frown \_ : (\operatorname{seq} X) \times (\operatorname{seq} X) \to (\operatorname{seq} X)$$

$$\forall\, \sigma, \tau : \operatorname{seq} X \bullet \sigma \frown \tau = \sigma \cup \{n : \operatorname{dom}\tau \bullet (n + \#\sigma) \mapsto \tau(n)\}$$

]

104. **Wie bestimmt man Head und Tail einer Sequenz?** [Mit den sogenannten Sequenz distruktoren:

$$[X]$$

$$head : \operatorname{seq} X \pfun X$$

$$tail : \operatorname{seq} X \pfun \operatorname{seq} X$$

$$head = \forall\, \sigma : \operatorname{seq} X \mid \sigma \neq \langle\rangle \bullet head\, \sigma = \sigma(1)$$

$$\forall\, \sigma : \operatorname{seq} X \mid \sigma \neq \langle\rangle \bullet \#tail\, \sigma = \#\sigma - 1 \;\; \forall\, i : 1 \mathinner{..} \#\sigma - 1 \bullet (tail\, \sigma)(i) = \sigma(i + 1)$$

Wenn dioe Sequenz nicht leer war, dann erhält man aus Head und Teil zusammen wieder die Sequenz]

105. **Wie filtert man in Z Elemente aus einer Sequenz heraus?** [$\sigma \upharpoonright F$ bedeutet, SIgma ist die geordnete Sequenz. Die Elemente von Sigma sind Elemente der Menge X. F ist nun eine Teilmenge von X, das heisst es werden nur die Elemente der Sequenz zurückgegeben (als Sequenz) die auch in F vorkommen. z.B.:
   - $\langle a, b, c, d, e, d, c, b, a\rangle \upharpoonright \{a, d\} = \langle a, d, d, a\rangle$
   - $\langle\rangle \upharpoonright F = \langle\rangle$
   - $\sigma \upharpoonright \varnothing = \langle\rangle$

   ]

106. **Wie extrahiert man bestimmte Elemente einer Sequenz in Z?** [$E \upharpoonright \sigma$ bedeutet, dass aus Sigma diejenigen Elemente ausgegeben werden, für die ein Index in E vorkommt, wobei E Teilmenge der Indexmenge N_1 ist (Menge der natürlichen Zahlen), z.B.:
   - $\{0, 1, 3, 5\} \upharpoonright \langle apr, jan, dec, sep \rangle = \langle apr, dec \rangle$
   - $E \upharpoonright \langle\rangle = \langle\rangle$]
107. **Was ist eine injektive endliche Sequenz?** [Eine Sequenz ohne Duplikate]
108. **Wie kehrt man eine Sequenz um?**
[

$$[X]$$
$$rev\_ : (\operatorname{seq} X) \rightarrow (\operatorname{seq} X)$$
$$\forall \sigma : \operatorname{seq} X \bullet rev\, \sigma = \lambda\, n : \operatorname{dom} \sigma \bullet \sigma(\#\sigma - n + 1)$$

]
109. **Was sind Multimengen (Bags)?** [Multimengen sind Mengen, in den Elemente mehrmals vorkommen können. Die Ordnung ist nicht wichtig. Multimengen sind geklammert und haben ein spezielle Elementsymbol, z.B.:
   - $[\![3, 5, 3, 1, 9]\!] = [\![1, 3, 3, 5, 9]\!] \neq [\![1, 3, 5, 9]\!]$
   - $3 \sqsubseteq [\![3, 5, 3, 1, 9]\!]$.

   Es gibt noch spezielle Operatoren für z.B. Vereinigung, Schnitt, Differenz und Auftretenshäufigkeitszähler ➔ Nicht besprochen]
110. **Aus welchen zwei Sub-Sprachen besteht Z?** [Mathematische Sprache, um Designaspekt, Objekte und deren Relationen zu modellieren| Schematasprache um Modellierunge (Daten, Funktionen und Prädikate) zu strukturieren, komponieren und zu Teilen)]
111. **Für was alles kann man nun Z Schemata verwenden?** [Als Deklarationen, Typen und als Prädikate benutzt werden| Man kann damit den Zustandsraum (Zustände und Zustandsübergänge) eines Systems modellieren| Sie können benutzt werden um formale Modellierungen letztendlich zu verifizieren]
112. **Was versteht man unter Horizontaler bzw. vertikaler Syntax?** [Vertikale Syntax:

name
declaration of typed variables (represent observations of the state)
relationships between values of vars (invariants of the system)

$SchemaOne$
$a : \mathbb{Z}$
$c : \mathbb{P}\mathbb{Z}$
$a \in c$
$c \neq \varnothing$

H

$$\text{name} \mathrel{\widehat=} [\text{declarations} \mid \text{predicate}]$$

orizontale Syntax:]
113. **Was sind „Tagged Record Types“ und „Binding“?** [Ein Record ist ein zusammengesetzter Datentyp

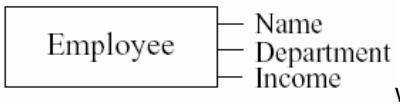
$SchemaTwo$
$a : \mathbb{Z}$
$c : \mathbb{P}\mathbb{Z}$

| Ein Binding ist dann eine Instanz eines Record, also eine Bindung zwischen Variablen eines Objektes und deren Inhalt, ausgedrückt wird dies z.B. durch:
$\langle\!|\ a \rightsquigarrow 2, c \rightsquigarrow \{1, 2, 3\}\ |\!\rangle$ Angesprochen werden die Variablen über den Selectoperator „_._“,d.h. wenn s ein das Objekt des obigen Schemas ist, dann kann man über s.a und s.c die Inhalte der beiden Variablen ansprechen]

114. **Was versteht man unter charakteristischer Bindung?** [Jede Komponente wird an einen Wert mit demselben Namen gebunden $\langle\!| \; c_0 \rightsquigarrow c_0, \ldots, c_n \rightsquigarrow c_n \; |\!\rangle$ =ΘA | ΘA bedeutet, dass es sich hierbei um die charakterische Bindung von Schema A handelt]

115. **Kann man ein E/R Diagramm in Z umwandeln und wenn ja wie?** [Ja, insbesondere können Constraints auf Daten formalisier werden, die in E/R Diagrammen implizit sind. Die Transformation geht in 3 Schritten:

    1. Übersetzung der Basisentitäten, also z.B.:

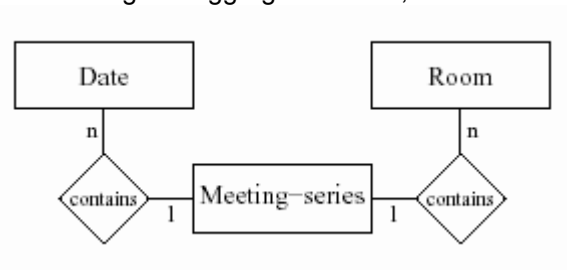


wird umgeformt in

$$
\begin{array}{|l}
\textit{Employee} \\
\hline
\textit{Name} : \textit{NAME} \\
\textit{Dept} : \textit{DEPARTMENT} \\
\textit{Income} : 1000 \mathinner{..} 5000 \\
\hline
\textbf{if } \textit{Dept} = A \textbf{ then} (1000 \leq \textit{Income}) \land (\textit{Income} \leq 3000) \\
\quad \textbf{else if } \textit{Dept} = B \textbf{ then} (3000 \leq \textit{Income}) \land (\textit{Income} \leq 5000) \\
\quad\quad \textbf{else} (4000 \leq \textit{Income}) \land (\textit{Income} \leq 6000)
\end{array}
$$

    ]

    2. Übersetzung der Aggregatentitäten, also z.B. :



wird umgeformt in

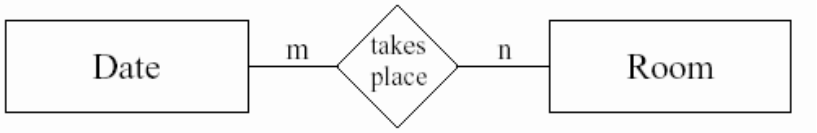
$$\textit{Meeting} - \textit{Series} == (\operatorname{seq} \textit{Date}) \times (\operatorname{seq} \textit{Room})$$

or

$$\textit{Meeting} - \textit{Series} == (\mathbb{P}\, \textit{Room}) \times (\mathbb{P}\, \textit{Date})$$

    ]

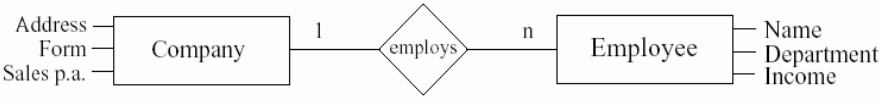
    3. Übersetzung der Relationen, also z.B.:



wird in die Relation

$$\textit{takes} - \textit{place} == \textit{Date} \leftrightarrow \textit{Room}$$

umgeformt und

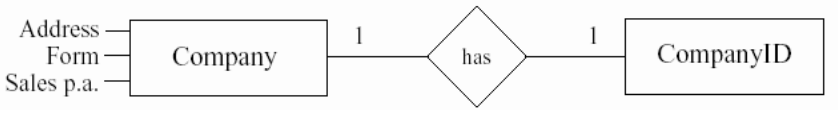


entspricht der Funktion

$$\textit{employs} == \textit{Employee} \nrightarrow \textit{Company}$$

    ]

    4. Was sollte man machen, wenn eine Entität einen Primärschlüssel enthält? [In diesem Fall ist eine Expansiion sinnvoll, z.B.:

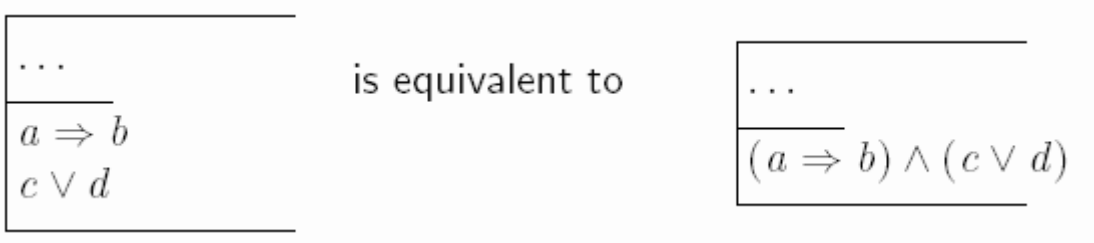


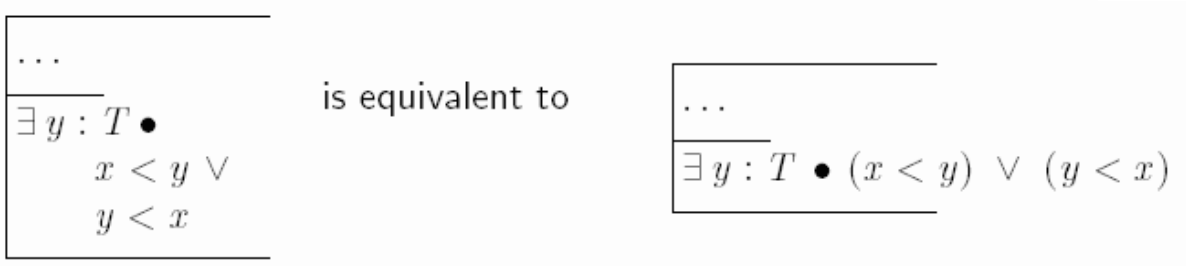
ergibt die partielle Injektion

$$has == CompanyId \rightarrowtail Company$$

]

116. **Wann nennt man zwei Schemata Äquivalent?** [Sie sind äquivalent, wenn sie die gleichen Variablen einführen und die gleichen Constraints auf sie setzen| Constraints können implizit bereits in der Deklaration enthalten sein! Z.B.:





]

117. **Nennen Sie ein Beispiel bei dem das Z-Schema als Deklaration verwendet wird!**

[

$$MONTHS ::= jan \mid feb \mid mar \mid apr \mid may \mid jun \mid jul \mid aug \mid sep \mid oct \mid nov \mid dec$$

Date

$$month : MONTHS$$
$$day : 1 \,..\, 31$$
$$month \in \{apr, jun, sep, nov\} \Rightarrow day \leq 30$$
$$month = feb \Rightarrow day \leq 29$$

Die Deklaration kann auch vor einem Quantor verwendet werden, für das obige Date Beispiel z.B.:

$$\mathcal{Q}\, Date \bullet p \quad \Leftrightarrow \quad \mathcal{Q}\, month : MONTHS,\ day : 1 \,..\, 31 \mid (month \in \{apr, jun, sep, nov\} \Rightarrow day \leq 30) \wedge (month = feb \Rightarrow day \leq 29) \bullet p$$

somit ist z.B.:

$$\exists\, Date \bullet (month = feb) \wedge (day = 29) \quad \text{is } true$$

$$\forall\, Date \bullet day \leq 30 \quad \text{is } false, \text{ as there is } \langle\!|\ month \rightsquigarrow mar,\ day \rightsquigarrow 31\ |\!\rangle.$$

]

118. **Wie und unter welchen Vorraussetzungen kann man ein Schema auch als Prädikat verwenden?** [Jede Komponente muss bereits als Variable des richtigen Typs deklariert worden sein. Beispiel:

$$\begin{array}{|l} SchemaOne \\ \hline a : \mathbb{Z} \\ c : \mathbb{P}\,\mathbb{Z} \\ \hline a \in c \ \land \ c \neq \varnothing \end{array} \quad \text{and} \quad \begin{array}{|l} SchemaThree \\ \hline a : \mathbb{Z} \\ c : \mathbb{P}\,\mathbb{Z} \\ \hline a \in c \ \land \ c \neq \varnothing \\ c \subseteq \{0, 1\} \end{array}$$

$$\forall\, a : \mathbb{Z};\ c : \mathbb{P}\,\mathbb{Z} \mid SchemaOne \bullet SchemaThree$$
$$\Leftrightarrow$$
$$\forall\, a : \mathbb{Z};\ c : \mathbb{P}\,\mathbb{Z} \mid (a \in c) \ \land \ (c \neq \varnothing) \bullet (a \in c) \ \land \ (c \neq \varnothing) \ \land \ (c \subseteq \{0, 1\})$$

Alle Variablen aus SchemaOne müssen auch die Bedingungen aus SchemaThree erfüllen]

119. **Was passiert, wenn der Deklarationsteil eines (Prädikat)-Schemas bereits Constraints enthält und der andere nicht?** [Dann muss man normalisieren, dafür gibt es eine kanonische Form, die die Constrainst aus dem Deklarationsteil in den Bedingungsteil zieht

$$\begin{array}{|l} SchemaFour \\ \hline a : \mathbb{N} \\ c : \mathbb{P}\,\mathbb{N} \\ \hline a \in c \ \land \ c \neq \varnothing \end{array} \quad \text{is not equivalent to} \quad \begin{array}{|l} SchemaOne \\ \hline a : \mathbb{Z} \\ c : \mathbb{P}\,\mathbb{Z} \\ \hline a \in c \ \land \ c \neq \varnothing \end{array}$$

SchemaFour wird dann normalisiert zu

$$\begin{array}{|l} SchemaFourNormalized \\ \hline a : \mathbb{Z} \\ c : \mathbb{P}\,\mathbb{Z} \\ \hline a \in \mathbb{N} \\ c \in \mathbb{P}\,\mathbb{N} \\ a \in c \ \land \ c \neq \varnothing \end{array}$$

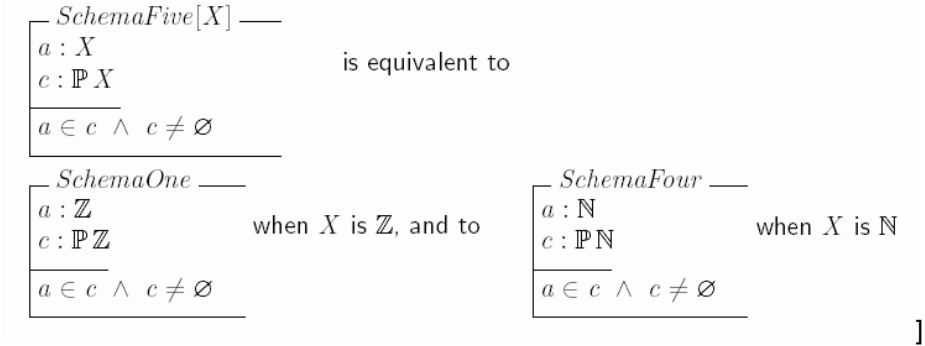
]

120. **Welche (logischen) Operationen gibt es auf Z-Schemata?** [Umbenennung, Inklusion, Dekoration, Konjunktion, Disjunktion, Normalisierung, Negation, Quantifikation, Hiding, Piping ➔ Wir können Spezifikationen (sequentiell) strukturieren und komponieren]
    a. **Was geschieht beim Umbenennen eines Schemas und wie lautet die Syntax?** [Komponenten(Variablen) werden umbenannt, die Syntax ist *Schema[new/old]*

$$\begin{array}{|l} SchemaOne \\ \hline a : \mathbb{Z} \\ c : \mathbb{P}\,\mathbb{Z} \\ \hline a \in c \ \land \ c \neq \varnothing \end{array} \quad \Longrightarrow \quad SchemaOne[q/a, s/c] \ \text{is equivalent to} \quad \begin{array}{|l} \hline q : \mathbb{Z} \\ s : \mathbb{P}\,\mathbb{Z} \\ \hline q \in s \ \land \ s \neq \varnothing \end{array}$$

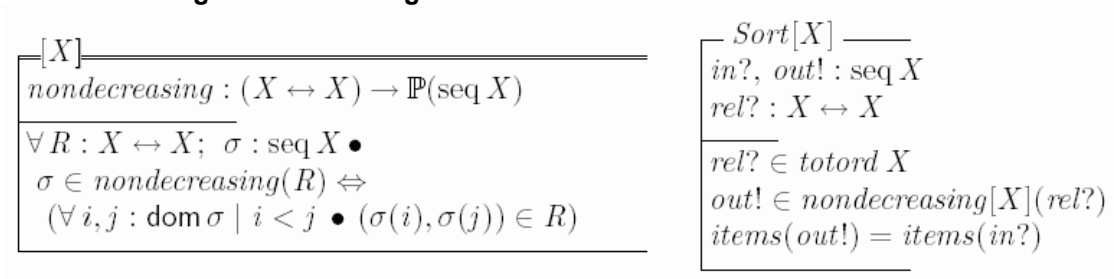
    ]
    b. **Die normale Umbenennung behält die Variablentypen bei, was aber, wenn man nur die Struktur beibehalten will, aber auch die Typen ändern will?** [Dann muss

man ein sogenanntes generisches Schema mit formalen Parametern erzeugen



]

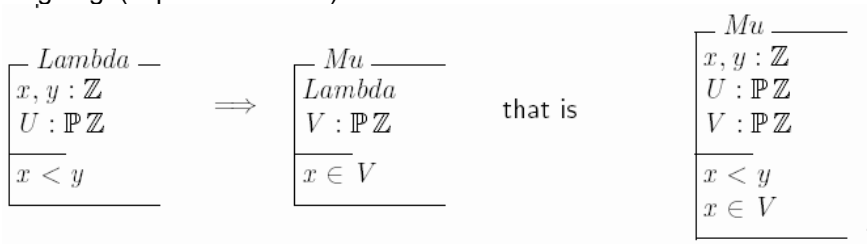
c. **Erklären sie folgenden Sortieralgorithmus!**



[Die Eingabe ist eine Sequenz|Die Ausgabe ist eine Sequenz sortiert im Bezug auf die eingegebene Relation| Links wird der Typ „nondecreasing“ deklariert, er enthält die Abbildung X auf X die in der Relation übergeben wurde. R ist eine Menge vom Typ X auf X die die Relation erfüllt. Die unterste Zeile links enthält die paarweise Abfrage der Elemente der Sequenz und überprüft die Bedingung auf die Menge R| Das Ergebnis steht dann am Ende in non-decreasing(R) drin und wird in „out!“ übergeben] (Etwas chaotisch erklärt…☺)

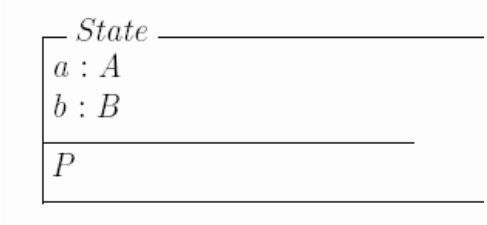
d. **Was muss man bei der Konjunktion zweier Z-Schemata beachten?** [Kommen Variablennamen in beiden Schemata vor, müssen sie vom gleichen Typ sein. Die Variablen werden dann alle zusammgeschmissen und die Bedingungen „ver-undet“ (A und B). Der Vorteil ist, dass man Teile eines Systems getrennt spezifizieren kann und erst später wieder zusammen bringt (Separation of Concerns)]

e. **Was bedeutet Inklusion bei Z-Schemata?** [Ein Schema wird in ein anderes eingefügt (importiert/vererbt)



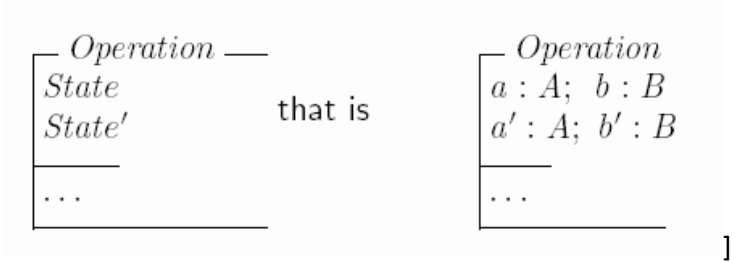
]

f. **Was ist Decoartion von Schemata?** [Hierbei geht es um Zustände, bzw. Zustandsübergänge| Ein Operation auf dem Zustand wird durch zwei Kopien von „State“ beschrieben, eine vor und eine nach der Operation (State und State’)



Das Prädikat P ist Zustandsinvariante| Eine

Operation sieht dann so aus:

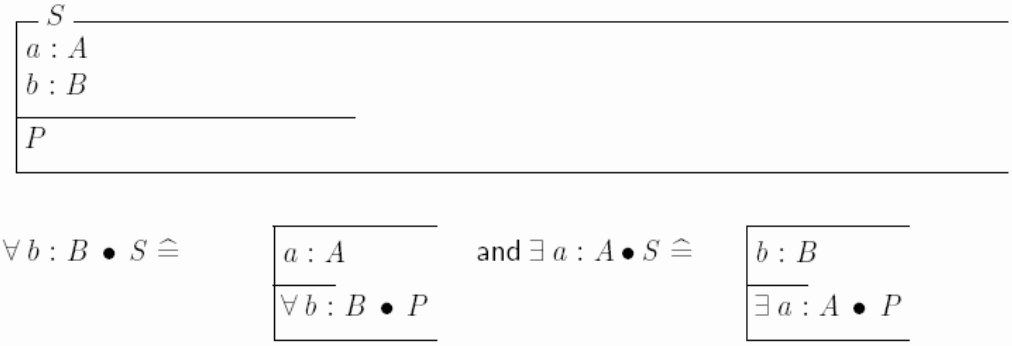


]

i. **Welche zwei Typen von Operationen gibt es?** [Es gibt Operationen die den Zustand verändern, markiert durch ein Δ vor den „importierten" Daten, und Operationen die den Zustand nicht verändern, markiert durch Ξ vor den „importierten" Daten]

g. **Wie funktioniert Disjunktion von Z-Schemata?** [Das gleiche Vorgehene wie bei der Konjunktion, nur das die Bedingungen ver-odert werden, somit kann man Alternativen im Verhalten eines Systems beschreiben]

121. **Was ist zu tun um ungültige Eingaben z.B. bei AddBirthday des BirthdayBook Beispiels zu verhindern?** [Man erstellt verschiedene Schemata die die möglchen auftretenden Fehlerfälle erkennen und abfangen – Diese verschiedenen Fehlerfälle kann man dann durch „Ver-Und-ung in die AddBirthday" Operation mit einbringen]
122. **Theroretisch könnte man über die o.g. Operationen ja eine komplettes Schema welches alle Eigenschaften vereint basteln, warum sollte man das nicht tun?** [Modularität geht dann vollständig verloren]
123. **Wie funktioniert die Negation von Schemata?** [Schema normalisieren soweit noch nicht der Fall, dann die Bedingung negieren]
124. **Was versteht man unter Quantifikation und Hiding?** [Es kann über einige Komponenten des Schemas quantifiziert werden. Hiding ist ein anderer Ausdruck für die existentielle Quantifizierung



]

125. **Was versteht man unter Komposition (Piping) von Z-Schemata**

$OpOne \fatsemi OpTwo$ **?** [Die Hintereinanderausführung zweier Operationen

$\Longrightarrow OpOne \fatsemi OpTwo = (OpOne[a''/a',\ b''/b'] \land OpTwo[a''/a,\ b''/b]) \setminus (a'', b'')$

]

126. **Was ist Haskell?** [Eine stark getypte funktionale Programmiersprache höherer Ordnung]
    a. **Was sind Klassen in Haskell?** [Sie werden benutzt zum Überladen von Operatoren und hierarchische Strukturierung]
    b. **Was ist ein Polymorphismums?** [Polymorphie nutzt man aus um Methoden auf unterschiedlich Daten anzuwenden. Die Wiederverwendbarkeit steht hier im Vordergrund]
    c. **Was ist ein parametrischer Polymorphismus?** [Eine allgemeine Implementierungdie für alle Typinstanzen funktioniert]
    d. **Funktionen abstrahieren berechnungsschritte in einem Programm, was ist das Analogon zu Funktionen in einer imperativen Programmiersprache?** [Prozeduren]
    e. **Was ist ein Modul?** [Ein Modul definiert eine Schnittstelle, in Haskell definiert ein Modul eine Sammlung von Werten, Datentypen, Typdefinitionen, Klassen usw. in einer Umgebung] Sie bieten Namensraummkontrolle an und können für abstrakte Datentypen benutzt werden.]

f. **Wie wird ein Module definiert?**

```
module Tree where

data Tree a = Leaf a | Node (Tree a) (Tree a)

fringe :: Tree a -> [a]
fringe (Leaf x) = [x]
fringe (Node l r) = fringe l ++ fringe r
```

[ ]

g. **Wie importiert man ein Modul in ein anderes?**

```
module foo where
import Tree

...  fringe ( ... Leaf ... ) ...
```

[ ]

h. **Wie funktioniert die Namensraumkontrolle?** [Die Variablen und Objekte werden explizit aufgezählt und damit kontrolliert]

127. **Was ist Daten-Typ und wie definiert man diesen?** [Ein Datentyp ist eine Datenmenge und die zugehörigen Funktionen. Signaturen beschreiben die Typen der

```
data Tree a     -- just the type name
leaf            :: a -> Tree a
node h          :: Tree a -> Tree a -> Tree a
cell            :: Tree a -> a
left,right      :: Tree a -> Tree a
isLeaf          :: Tree a -> Bool
```

Funktionen.]

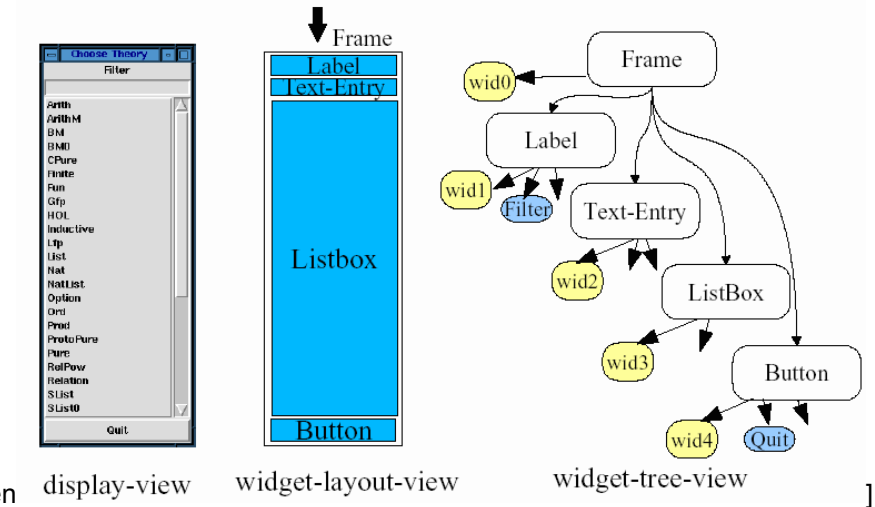
a. **Was nun ein abstrakter Datentyp?** [Ein Datentyp dessen Implementierung gekapselt (Änderungen immer lokal) ist. Den Zugang zu den Daten hat man nur durch die Funktionen in der Signatur. Module sind eine Möglichkeit ADTs zu implementieren]

128. **Was ist SML?** [„Structured Modelling Language“. Wie Haskell eine stark getypte Programmiersprache höherer Ordnung]
    a. **Was sind die Hauptunterschiede zu Haskell?** [SML besitzt Referenztypen und ein sehr fortgeschrittenes Modulsystem]
    b. **Was können Module in SML?** [Sie unterstützen Kapselung, trennen Schnittstellenspezifikationen (Signature) von Implementierung (Struktor vs. Funktor), Module können parametrisiert sein (Polymorphe Typen), Funktoren der Module unterstützen die Zusammensetzung von Datentypen, Separate Compilierung einzelner Module möglich]
129. **Welche Strukturierungsmöglichkeiten bieten Objekt orientierte Programmiersprachen an?** [Klassen, Schnittstellen (Interfaces), Vererbung, Polymorphismus. Sie unterstützen sowohl klassische ADTs als auch Objekt orientierte Entwicklung]
130. **Wie wird die Namensraumkontrolle in Java geregelt?** [Es lässt sich für jede Klasse, Methode, Variable angeben ob sie public, protected oder private sein soll (Public bedeutet: dass man z.B. auf Variablen eines Objektes von aussen zugreifen kann| Private bedeutet: man kann von ausserhalb der Klasse/Objekt nicht auf Variablen zugreifen, aus einer abgeleiteten Klasse/Objekt geht dies jedoch| Protected ist ganz restriktiv, es gibt keine Möglichkeit weder von außen noch aus einer abgeleiteten Klasse auf eine Variabel einer Klasse/Objekt zuzugreifen)]
131. **Was bedeutet Kapselung?** [Ein Objekt kann abgekapselt in z.B. einem separaten Paket enthalten sein, der Objektcode ist abgekapselt vom Programmcode|Public,Private und Protected spezifiert weitere Kontrolle]
132. **Was ist ein Interface?** [Eine Definition wie Objekte miteinander interagieren können. Ein Interface legt sowohl syntaktische als auch semantische Eigenschaften fest, d.h. es legt eigentlich einen Kommunikationsvertrag fest]
133. **Was ist der Unterschied zwischen Polymorphismus bei der OO-Programmierung und parametrischem Polymorphismus in Haskell und SML?** [Mehrere Klassen können eine Schnittstelle implementieren| Interface kann als Typ benutzt werden]

134. **Welche zwei Arten von Vererbung gibt es?** [Vererbung von Implementierung (Die Deklaration von Subklassen ermöglicht die Wiederverwendbarkeit von Implementierungen|Vererbung von Interface➔ Java unterstützt einfache Vererbung von Implementierungen (andere OO-Sprachen erlauben dies unter Benutzung komplizierter Präzidenzregeln um Mehrdeutigkeiten zu eliminieren) und mehrfache Vererbung von Interfaces]
135. **Nennen Sie Vor- und Nachteile von Vererbung?** [Vorteile: Das Interface wirkt als Vertrag, Kapselung, Code-Wiederverwendbarkeit/ Modifizierbarkeit|Unterstützung von Frameworks und andere OO-Entwurfs Prinzipien| Nachteile: Explosion von Klassen|Geschwindigkeit ist schlechter (Kosten von Late-Binding)| Programmkomplexität nimmt zu (Modelle können hier als Hilfsmittel dienen)]
136. **Wie machen wir den Übergang vom Entwurf zum Code?** [Modelle müssen schrittweise verfeinert werden| Zusätzlich Informationen über statische oder dynamische Eigenschaften werden hinzugefügt| Auch möglich die Refaktorisierung wobei Verantwortungen umverteilt werden, z.B. die Einführung von Designpattern oder die Benutzung vorgefertigter Komponenten (Bibliotheken)| Aber: Die Synthese zwischen Spezifikation und Implementation ist ein Problem für sich, man muss Verfeinerungen vornehmen, algorithmische Entwicklung usw. ➔ Das Problem der Synthese ist unentscheidbar!!!]
137. **Wo fängt man an?** [Der Ausgangspunkt bei uns ist entweder ein Modell in UML modelliert oder eine Z-Spezifikation➔ Manche Dinge sind dabei recht einfach zu übersetzen so z.B. ein Klassen Diagramm (Class Diagram) in eine Java Klassenhierarchie (Class Hierarchy)| Andere Aspekte sind hingegen schwerer, z:b. die Synthese von Sequenz Diagrammen| Bestimmte Umwandlungen können evtl. direkt vom Case Tool übernommen, allerdings typischerweise nur die Übersetzung Klassendiagramm nach Java Klassenhierarchie]
138. **Warum sind gerade Klassendiagramme besonders gut geeignet zur Codegenerierung?** [Statische Constraints haben eine relativ einfache Semantik| Interpretation wird oft eindeutig bestimmt, z.B. A ist eine Subklasse von B. Diagrammkomponenten spiegeln Aspekte von OO Sprachen wieder]
139. **Was ist bei Klassendiagramm eher problematisch?** [Funktionale und dynamische Aspekte sind schwerer. Spezifikationen sind nicht konstruktiv. Funktionalität ist oft nicht ausreichend definiert (z.B. Message Sequenzen ➔ Datenaustausch). Eindeutige, festgelegte Funktionen können auf unterschiedliche Weise implementiert werden]
140. **Was versteht man unter Verfeinerung?** [Verfeinerung behandelt die Umwandlung einer abstrakten Spezifikation in eine konkrete. Verfeinerung führt zu einem stärkeren, festgelegteren Modell| Semantisch gesehen ist das Modell deterministischer| Logisch gesehen folgt daraus die abstrakte Spezifikation➔ Natürlich kann man die Korrektheit einer solchen Verfeinerung nachweisen, also dass unter der Annahme einer Invarianten konkrete Implementierung stärker ist als die Abstrakte]
141. **Wie implementiert man Z-Spezifikationen die Mengen enthalten in einer Sprache mit Arrays und Listen?** [Das eigentlich Problem ist, dass eine Programmiersprache nicht direkt mit Mengen umgehen kann, deshalb verfeinert man eine Mengenspezifikation zu Sequenzen, denn Sequenzen können später in der Zielsprache direkt implementiert werden]
142. **Wie überprüft man die Korrektheit mathematisch logisch?** [Man nimmt den Antezedens (Vorbedingungen) und versucht durch Umwandlungen mit Hilfe der einzelnen Vorbedingungen über einen Äquivalenzbeweis auf die Konsequenz (Verfeinerung) zu kommen, gelingt dies, ist die Verfeinerung korrekt]
143. **Welche Idee gibt es Z-Spezifikation auf eine Programmiersprache abzubilden?** [Für die Z Typen werden Datentypen in der Programmiersprache definiert, axiomatische Definitionen werden in Konstanten oder nicht sehr veränderbare Daten transformiert. Schemata werden in veränderbare Daten überführt]
144. **In wie fern können Case-Tools einem die Implementationsarbeit abnehmen?** [Sie erzeugen lediglich ein Skelett ➔ Kreativität kann nicht durch ein Tool ersetzt werden| Vorraussetzung ist auf jeden Fall reine formale Spezifikationssprache wie Z – Z gibt es auch in Objekt orientierten Varianten die die OO Features und auch die Kontrolle besser unterstützen]
145. **Was sind Design Pattterns und für was braucht man diese?** [Unser Problem ist dass High Level Modelle zu abstrakt sind und implentierungsdetails unterschlagen. Design Models hingegen können zu problemspezifisch sein, eine Lösung des Problems ist es z.B. , das Modell mit zusätzlichen Details zu verfeinern. U.A. dies wird durch die Design Patterns realisiert.]

a. **Was ist die Grundidee von Design Patterns?** [Das Ziel ist es eine allgemeine Lösung für eine Klasse von Entwicklungsproblemen zu beschreiben, jedes Pattern beschreibt dann also ein Problem, welches immer und immer wieder auftritt und es beschreibt den Kern Lösung des Problems und zwar so, dass man die Lösung millionenfach verwenden kann ohne den Lösungsweg zweimal gehen zu müssen.Die so genannte „Gang-Of-Four bestehend aus 4 Wissenschaftlern entwickelten einen Design-Pattern Katalog, der Expertenwissen über Designprinzipien um die Qualität von Software entscheidend zu verbessern]
b. **Welchen Nutzen kann man aus Design Patterns ziehen?** [Einfachere Entwicklung, höhere Wiederverwendbarkeit, Dokumentation]
c. **Welche Informationen stecken in einem Design-Pattern?** [Name, Type, Intent, „Also Known as“, Motivation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation]

146. **Nennen Sie die wichtigsten Design-Ziele einer GUI, wie sie auch Schneidermann präzisiert hat!** [
    1. Reduzierung der Information die das Kurzzeitgedächtnis zu verarbeiten hat
    2. Visuelle Repräsentation von Objekt
    3. Adaptiv sowohl für Novizen als auch für Experten
    4. Konsistente Darstellung und Gestaltung
    5. Konform zu den Erwartungen der Benutzer
    6. Informatives Feedback und einfache und verständliche Fehlerbehandlung
    7. Einfaches rückgängigmachen von Aktionen
    8. Dialogfenster entwerfen um das System in sich geschlossen zu halten]
147. **Woran würden Sie sich orientieren, wenn Sie eine GUI entwickeln sollten?** [Es gibt bereits schön gelöste Vorlagen, die aber speziell auf Spiele oder spezielle Graphikprogramme zugeschnitten sind (Kais Power Tools). Auch beim Webdesign gibt es viele gute Vorlagen an denen man sich orientieren kann. Letztendlich gibt es aber auch die GUI Style Guides, die einfach ein bestimmtes „Look and Feel“ in die GUI mit einbringen so z.B. „Mac“, „Motif“, „Windows“, „XP“]
148. **Was versteht man unter dem „Wimp-Style Design“?** [Windows, Icons, Menues, Pointers bzw. Widgets, Icons, Mice und Pull-Down Menues➔ Organisiert werden sowohl die Datenstruktur selbst als auch die graphischen Komponenten in einer Baumartigen Struktur, dies ist die Kernidee des



Ganzen]
149. **GUIs sind reaktive Systeme, was bedeutet das?** [Dass Ereignisse (wie Mouseklick oder Scroll) an Aktionen (z.B. Speichern oder öffnen eines Fensters) gebunden werden]
    a. **Bedingt durch das Event/Action System ergeben sich auch Probleme. Welche?** [Verschachtelungen, Sperren von Ressourcen (z.B. beim Datei-Öffnen ist der Editor selber gesperrt), Synchronisationen (z.B. Aktualisierungen von Vorlagen bei Word etc.), Sicherheitsabfragen/Meldungen etc.(z.B. „Wollen sie wirklich Schließen“)➔ Gerade durch diese Probleme wird das Entwickeln von GUIs oft unterschätzt]
150. **Mit welcher Darstellung modelliert man üblicherweise GUIs?** [Mit Event-Modellierungs Formalismen wie den State-Charts]

151. **Wie bestimmt man die System-Intergrationsgüte einer GUIs?** [Es gibt verschiedene Maße für Ergonomie und verschiedene Kriterien dazu. Beispielsweise das GOMS Modell]
152. **Wie bestimmt man die Softwareintegrationsgüte einer GUI (Validation)?** [Ganz einfach von Hand durchprobieren, oder durch immerwiederkehrendes Testen bestimmter Funktionen (auch durch generierte replay-Scripts möglich➔Pfad aus dem State-Chart-Diagramm angewandt entweder auf eine Dummy Datenmodell oder auf ein bereits realisiertes Datenmodell).|Formale Verifikation]
153. **Man kann Software aber auch formal validieren, wie?** [Formalisierung des GUI Event-Modells z.B. mit Temporal Logik, CSP| Formalisieren der Designziele| Verfeinerungen des GUI-Event Modells überprüfen]
154. **Können sie ein bisschen erklären wie das mit CSP geht?** [CSP ist eine Beschreibungssprache für Events/Actions. Es gibt z.B. Basissets für bestimmte Events, bestehend aus den jeweiligen Widgets und den zugehörigen „Events“ und „Actions“. Damit kann man das Verhalten eines Fensters beschreiben. Dadurch hat man eine sehr gute Möglichkeit eine formale Analyse darauf durchzuführen➔ z.B. Erkennen von Deadlocks| Es gibt auch Tools (z.B. FDR) die das automatisch machen]
155. **Was ist Java?** [Populäre Objektorientierte Programmiersprache, oo-Bibliotheksdesign, Sehr gut für konkurrierende Operationen (durch Threads), Unterstützung einheitlicher Designs (Metal, Motif, Windows)| High-Level Komponenten wie Treelists, Tables etc.]
    a. **Welche 2 Stilrichtungen gibt es bei Java in Bezug auf GUIs?** [Java/AWT: Peer-Approach (d.h. zugeschnitten auf System)➔ Probleme bei der Portabilität/Übertragbarkeit auf andere Systeme| JAVA/Swing: Painting-Approach (d.h. GUI wird direkt „gezeichnet“) ➔ System langsam| Das heisst Swing verwenden, das Ereignismodell von AWT wird bei Swing verwendet.]
    b. **Was können Sie noch zu Java/Swing sagen?** [Komponenten sind in Klassenhierarchie organisiert| Methoden werden im s.g. Modell-View-Controller mittels Designpatterns (d.h. Inhalt, visuelle Darstellung und Verhalten) organisiert]
    c. **Was ist oft schwierig zu lösen?** [Ansicht und Kontrolle deutlich zu trennen. Das ist bereits in sehr einfachen Fällen sehr schwer für den Programmierer]
    d. **Wie funktioniert Layout Management in Java/Swing?** [Es gibt einen Layoutmanager der das Layout kontrolliert. Es gibt z.B. ein Borderlayout, Flowlayout, Gridlayout…]
    e. **Wie funktioniert das mit dem Event-Modell in Java/Swing?** [Benutzt wird das Event-Modell von AWT| Den einzelnen Komponenten der GUI werden „Listener“ wie z.b. MouseListener, hinzugefügt, die auf bestimmte „Events“ wie z.B. MouseEvent reagieren]
    f. **Was sind Java Threads und für was braucht man sie?** [Präemptives Multitasking, d.h. mehrere Programminstanzen laufen (scheinbar) gleichzeitig ab. Jedem Thread wird eine gewisse Rechenzeit zur Verfügung gestellt. Jeder Thread wird dann nach und nach immer wieder kurze Zeit bearbeitet. Für Treads gibt es entsprechend verschiedene Aufrufe wie z.B. Control Priorities, Create Thread, Kill Thread, Group Threads| Locking von Objects oder Methoden ist problematisch, wenn es immer mehr und mehr davon gibt sinkt die Effektivität.]
156. **Was ist SmITK?** [Erweiterung von SML durch die Funtkionen von Tk Toolkit ➔ Tk Toolkit bringt Funktionen für z.B. Warnungsmeldungen, Filemanagement und Drag&Drop mit ➔ Anpassung von SML an das Window Konzept mit Widgets, Binding und Events|Schöne, kompakte Darstellung im SML Code]
157. **Was ist eine Middleware?** [Sie ermöglicht es Objekte auf einem Server vom Client aus direkt zu referenzieren. Es stellt also eine universale Kommunikationsplattformdar, die sowohl zugriffs- als auch ortstransparent ist. Sie stellt verschiedene Dienste zur Verfügung und setzt das OO Konzept zur Kapselung verschiedener Systeme um. Beispiele sind z.B. COM+,/.NET, EJB, CORBA]
158. **Was umfasst nun CORBA (Common Object Request Broker Architecture)?** [Sie ist eine Middleware, die plattform und sprachunabhängig ist. Corba ist ein Standard, bietet verschiedene Services und Möglichkeiten. Corba ist ein selbstberschreibendes System. Methoden können dynamisch ausgelöst werden]
    a. **Was ist die IDL (Interface Definition Language)?** [Eine Sprache (keine Programmiersprache) zur Beschreibung von Objekten mit welcher CORBA seine Daten zwischen Client und Server austauscht. IDL hat eine Trennung von Schnittstelle und Implementierung| Die Syntax ähnelt C++/Java| Keine Kontrollmechanismen| IDL unterstützt: Module, multiple Vererbung, Array und Sequenzen, Exceptions, Typen: Basistypen, struct, enum, typedef etc.]

b. **Was ist ein ORB und welche Aufgaben hat er?** [Ein Object Bus welcher aber die Details versteckt. Seine Aufgaben sind: Registrierung von Objektanforderungen; Finden von Objekten; Strukturieren von Parametern, Methodenauslösung; Strukurieren/Ordnen der Ergebnisse]
c. **Es gibt sowohl statische als auch dynamische Aufrufe, was sind die jeweiligen Eigenschaften?** [Beim statischen Aufruf SII (Static Invocation Interface) über Stubs und Skeletons ist strukturierender Code enthalten, Wir haben statisches Typechecking, alles ist zur compilierungszeit bekannt. Das statische Verfahren ist einfach und effizient| Beim dynamischen Aufruf über das DII (Dynamic Invocation Interface) wird die Anfrage zur Laufzeit konstruiert, die Metadaten stehen im Interface Repository, Typechecking erfoplgt nur über die Metadaten. Das dynamische Verfahren ist komplex aber sehr flexibel]

159. **Welche Hauptfragen stehen hinter der Validadierung von Software?** [Entwickelt man das richtige Produkt? Entwickelt man das Produkt richtig?]
160. **Was kann alles validiert werden?** [Eine fertige Implementation einer Software| Die Güte der Integration einer Software in ein bestehendes Computersystem| Die Güte der Integration in eine bestehende Arbeitsumgebung]
161. **Welche 3 Dimensionen werden bei der Validierung miteinbezogen?** [
    1. Validierungstechniken, d.h. Verifikation(1), Inspektion(2) und testing(3),…
    2. Validierungsstufen, d.h. Systemintegration(1), Softwareintegration(2), Implementation(3)
    3. Ziele der Validierung (was wird validiert): GUI(1), Funktionskernel(2), Interaktion der Komponenten(3), Interaktion des Systems mit der Umgebung(4)
    4. Die einzelnen Elemente der 3 Dimensionen sind dann jeweils kombinierbar, aber nicht jede Kombination macht Sinn.]
162. **Wie kann man validieren bei gegebenem Programm und gegebener Spezifikation(Post-Verification Approach)?** [Die Idee ist es logische Regeln anzuwenden um Äquivalenz zwischen der formalen Spezifikation und der Implementation zu zeigen. Dies kann Schritt für Schritt mit irgendeiner Art von Logic (z.B. Hoare-Logic, PL1, HOL, Z…)]
163. **Eigentlich könnte man ja auch die komplette Verfeinerungskette die entstanden ist verifizieren, wie stellt man hier die Verifikation sicher (Development by Refinement)?** [Man muss bei jedem Verfeinerungsschritt sicherstellen, dass die Verfeinerung auch sicher eine gültige Verfeinerung der vorhergehenden Spezifikation ist Für die letzte Verfeinerung kann dann Code generiert werden, von dem man sicher weiss, dass er korrekt ist. Der vergleich zweier Verfeinerungen erfolgt über diverse Logiktools wie Atelier B(Z-Logic), KIV (Dynamische Logik) etc.]
164. **Was sind die Vor- und Nachteile einer formalem Verifikation bzw. Transformation?** [Vorteil: Im Prinzip sehr gut anwendbar auf Software-Integration, die Software-Architektur und auch die enthaltenen Algorithmen. Theoretisch kann man damit den höchsten Grad an Qualität eines Software-Systems erreichen| Nachteile: Sehr, sehr teuer, da Hochqualifizierte Entwickler mitarbeiten müssen| keine Automatisierung bislang möglich (großes Forschungsgebiet| Es ist schwer eine komplette Softwareverifikation in den Softwareentwicklungsprozess einzubinden| Wird üblicherweise nur bei kritischen Aufträgen und sicherheitsrelevanten Entwicklungen angewandt]
165. **Man könnte doch auch gleich bei den einzelnen Verfeinerungsschritten eine korrekte Umwandlung sicherstellen. Sagen sie dazu mal was! (Transformational Approach)** [Wenn man ausgehende von einer ersten, recht einfachen Spezifikation immer nur durch Transformationsregeln (wie z.B. D&C) verfeinert, entstehen eigentlich keine Fehler, dies kann Interaktiv für jede neue Verfeinerung passieren. Man muss dann nur aus der letzten Verfeinerung den Code (automatisch) generieren]
166. **Welche Idee steckt hinter der Inspektion von Code?** [Das Programm von einem Dritten inspizieren lassen. Hierzu gibt es Checklisten für die einzelnen wichtigen Bereiche eines Programms (z.B.: Daten Referenzen, Deklarationen, Berechnungen, Vergleiche, Kontrollfluss, Schnittstellen, E/A…). Das Programmierteam veröffentlich ein Release ihres Programms. Eine Gruppe von Reviewern schaut den Code Zeile für Zeile durch und überprüfen verschiedene Aspekte des Systems. Gefundene Fehler werden in gemeinsamen Treffen besprochen (aber nicht gelöst). Es entsteht eine Fehlerprotokoll. Das Programmierteam korrigiert die Fehler an Hand des Protokolles. Die Inspektionen müssen gut geplant sein und das Reviewer Team muss unabhängig sein (Egocentric Bias)]

167. **Wie gut ist Validierung durch Inspektion?** [Man muss auf jeden Fall psychologische Faktoren beachten und die Erfahrung der Inspekteure| Kosten belaufen sich auf 15-20% der Entwicklungskosten, aber nur 60-70% der Fehler können gefunden werden. ABER: Trotzdem lohnende Investition||Probleme: Ergebnis sind Subjektiv und hängen z.T. von der Tagesform der Inspekteure ab| Lässt sich nicht auf große System skalieren. Relativ teuer| Inspektionen können Verbesserung behindern (d.h. z.B. eine mögliche noch in der Entwicklungsphase stehende Innovation die scheinbar schlecht ist wird möglicherweise verworfen.]
168. **Könnte man so eine Inspektion nicht auch automatisieren?** [Doch, ansatzweise. Man kann statische Analysetechniken verwenden um gefährliche/schlechte Konstrukte zu finden wie z.B. Typechecking, Konformität zur Dokumentation/Guidelines überpüfen, Check von unnötigem Kontroll-/Datenfluss, Überprüfung anhand von vorgegeben Software-Metriken.]
169. **Wie gut ist die automatische Inspektion? Vor-Nachteile?** [Vorteile: Automatisch, kann von Programmierern direkt benutzt werden| Nachteile: Überprüft nicht die Semantik eines Programms (Nur Typechecking reicht nicht aus)| Software-Metriken sind nicht einheitlich (Kontroverse Ansätze)]
170. **Was können Sie zum Testen von Software sagen?** [Testen ist die am Häufigsten verwendete Methode um Qualität von Software zu sichern, aber es wird oft recht unsystematisch angewandt, Testen ist sehr teuer (50% der Entwicklungskosten und 50% der Entwicklungszeit)| Tests ergänzen formale Spezifikationen sehr gut, neue Ideen können schnell ausprobiert (getestet) werden. Wenn die Spezifikation das falsche modelliert, dann findet man das durch Testen u.U. heraus ➔ Testen ist also keine Ersetzung von Verifikation, man kann durch Testen zwar wunderbar Fehler finden, aber was ist mit den Fehlern die nicht durch testen gefunden werden. Mit dem testen kann man auf jeden Fall die Zuverlässigkeit erhöhen, wenn auch die verwendeten Heuristikien z.T. undurchsichtig sind]
171. **Welche Testmethoden gibt es für die einzelnen Validierungsstufen?** [Akzeptanztest: usability, Alpha-/Betatest| Integrationstest: Interfaces und Module testen. Testen der Interaktionen von Komponenten (Vorbedingungen für Subkomponenten müssen erfüllt sein| Unittest: Testen einzelner Module oder Funktionen unter normaler Anwendung sowie unter Bedingungen die Fehler erzeugen müssten]
172. **Welche beiden Ansätze gibt es beim Integrationstest?** [Es gibt einerseits den Init-Access-Approach, welcher Zugang zu allen internen Zuständen benötigt, was aber nicht immer möglich und auch nicht immer erstrebenswert. Trotzdem ist dieser Ansatz vergleichweise einfach. Andererseits gibt es den Sequence-Test-Approach, welcher kein internes Wissen über Module etc. braucht aber sehr viel schwerer auszuführen ist.]
173. **Die wichtigste Frage ist ob ein Testkriterium überhaupt angemessen war für den jeweiligen Bereich. Welche Kriterien muss man ansetzen zum Testen der Struktur eines Programms oder einer Spezifikation über deren Kontrollfluss?** [Abdeckung aller Aussagen, Verzweigungen und Pfade durch Logik| Das „Cyclomatic Number Criterion“ von McCabe| Abdeckung multipler Bedingungen]
174. **Wie gut ist das?** [Es geht vollautomatisch, benutzt wohlbekannte Techniken aus dem Compilerbau]
175. **Welche Kriterien muss man ansetzen zum Testen der Struktur eines Programms oder einer Spezifikation über deren Datenfluss?** [Analog zum Kontrollflussgraph. Kriterien: Alle Definitionen, alle Verwendungen, Datenfluss zwischen den Prozeduren, Betrachtung der Daten im Kontext, Abhängigkeiten abdecken]
176. **Wie gut ist das?** [geht automatisch, Check auch von versteckten Zuständen möglich| Probleme mit Aliase und Pointern, aber eigentlich keine praktische Relevanz nur bei C++]
177. **Was ist Fault-Based Testing? Welche Kriterien sind hier relevant?** [Bewusstes Einbringen von Fehlern ➔ Erzeugen von Mutanten ➔ Vergleichkriterien zum „Richtigen“| Erkennen von Störungen, Vorhersage von Störungen]
178. **Wie gut ist Fault-Based-Testing?** [Gut zu verstehende semantische Basis| Unter bestimmten Bedingungen garantierte Fehlererkennung| Würde gut in die konventionelle Softwareproduktion passen| Keine formalen Spezifikationen notwendig| Guter Mittelweg um Fehler zu finden| Scheint automatisch zu funktionieren| Mutanten können über transformationen erzeugt werden]
179. **Welche Probleme gibt es beim Fault-Based-Testing?** [Viele ähnliche Operationen die sehr „Mannintensiv“ und damit teuer sind| Es ist schwierig ein gutes Errormodell für Software zu finden (was sind gute Errors)| Benötigt viel Rechenpower| geht bisher nur im Bereich Hardwaretesting| Insgesamt sehr begrenztes Anwendungsgebiet]

180. **Was ist die Hauptidee des „Specification-Based Input Space Partitioning“?** [Die Hauptidee ist es Bereiche des Input/Output Verhalten in Subdomains aufzuteilen] (SEHR KNAPP AUF DER FOLIE ➔ UNKLAR)
181. **Wie ist das beim Programm-Based Space Partitioning?** [Eingabe von Test-Cases entlang den Pfaden eines Kontrollflussgraphen um die Ausführung symbolisch zu simulieren] (SEHR KNAPP AUF DER FOLIE ➔ UNKLAR)
182. **Mit welcher Vorgehensweise kann man einen Unit-Test systematisch oder sogar automatisiert durchführen?** [Zunächste muss man das unendlich durch das endliche approximieren. Dazu gibt es zwei Hypothesen die dies konzeptualisieren. Die Uniformitätshypothese und die Regularitätshypothese. Auf diesen Hypothesen aufbauen wird die DNF-Methode angewandt, die einem sehr gut verschiedene Test-Cases liefern kann. Dies geht durchaus automatisiert]
183. **Was besagt die die Uniformitäts Hypothese?** [Dass sich ein System bei einigen (endlich vielen) Sets aus einer Gesamtmenge von unendlichen vielen Sets von Eingaben einheitlich verhält. D.h. es gibt nur endlich viele Sets für die sich das System einheitlich verhält. Wir können dann aus diesen Sets direkt bestimmte Repräsentative testen ohne das gesamte Set testen zu müssen. (Man muss in einer Kiste mit gleichen Äpfeln ja auch nicht jeden anbeissen um zu wissen wie sie schmecken)

$$\frac{\bigcup_{i:1..n} S(i) = A \wedge \quad \forall\, i:1..n \bullet \exists\, x:S(i) \bullet P(x)}{\forall\, t:A \bullet P(t)}$$

]
184. **Was besagt die Regularitätshypothese?** [Man nimmt an, dass jede Eingabe t mit einer bestimmten Komplexität |t|, die kleiner ist als eine Grenze n, ausreicht um das korrekte Verhalten des Systems nachzuweisen.

$$\frac{\forall\, t \bullet |t| < n \Rightarrow P(t)}{\forall\, t \bullet P(t)}$$

]
185. **Was versteht man unter der DNF-Methode?** [Durch die Uniformitätshypothese kommt man auf die Idee die einzelnen Partitionen über Disjunktive-Normalformen zu berechnen:

$$\forall\, x_1, \ldots, x_n \bullet D_1(x_1,..,x_n) \vee \ldots \vee D_m(x_1,..,x_n)$$

Man muss die einzelnen Argumente der D_i jetzt noch so wählen, dass die Disjunktion wahr wird]

186. **Nenne ein Beispiel anhand dessen Sie die DNF-Methode genauer erläutern können?**

$$\text{triangle} : \mathbb{P}(\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z})$$

$$\forall\, x, y, z : \mathbb{Z} \bullet \text{triangle}(x, y, z) = x>0 \land y>0 \land z>0 \land x+y > z \land y+z > x \land x+z > y$$

$$\text{res} ::= \text{equilateral} \mid \text{isosceles} \mid \text{scalene} \mid \text{error}$$

$$\text{program} : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \Downarrow \text{res}$$

$$\forall\, x, y, z : \mathbb{Z} \bullet \text{triangle}(x, y, z) \land \text{program}(x,y,z) = \text{if } x=y \text{ then if } y=z \text{ then equilateral else isosceles else if } y=z \text{ then isosceles else if } x=z \text{ then isosceles else scalene} \lor \neg\text{triangle}(x, y, z) \land \text{program}(x,y,z) = \text{error}$$

[
Bestimmte Partitionen von Eingaben ergeben nun ganz bestimmte Ausgaben, nämlich „gleichseitig“, „gleichschenklig“,,ungleichseitig“ „fehlerhaft“.Die daraus entstehenden Regeln kann man in Isabelle/HOL) schreiben:

$$x = y \land y = z \land \text{triangle}(z, z, z) \land \text{program}(z, z, z) = \text{equilateral} \lor$$
$$x = y \land y \neq z \land \text{triangle}(y, y, z) \land \text{program}(y, y, z) = \text{isosceles} \lor$$
$$y = z \land x \neq z \land \text{triangle}(x, z, z) \land \text{program}(x, z, z) = \text{isosceles} \lor$$
$$x = z \land y \neq z \land \text{triangle}(z, y, z) \land \text{program}(z, y, z) = \text{isosceles} \lor$$
$$x \neq y \land x \neq z \land y \neq z \land \text{triangle}(x, y, z) \land \text{program}(x, y, z) = \text{scalene} \lor$$
$$\neg\, \text{triangle}(x, y, z) \land \text{program}(x, y, z) = \text{error}$$

Nun wird für jede der obigen Aussagen eine Lösung generiert um die Test cases zu erhalten

Normal Behaviour

x	y	z	program
2	2	2	equilateral
1	1	2	isoscele
2	1	1	isoscele
1	2	1	isoscele
4	5	3	scalene

Exceptional Behaviour

x	y	z	program
-2	2	2	error
1	-2	2	error
2	1	-1	error
1	2	4	error
4	1	2	error
1	4	2	error

]

187. **Was kann die DNF Methode nun also?** [Prinzipiell automatische Generation von Test-Classes, wenn die Spezifikation nur einen ganz aussen stehenden All-Quantor besitzt| Erlaubt die Veränderung von Spezifikationen| Benötigt das Beweisen von Theorem| Produziert ein minimales Test-Set| Es kann passieren, dass sich das Problem signifikant aufbläht| Generierung von Test-Sets ist theoretisch einfach, aber technologisch sehr schwer zu lösen (BDDs)]
188. **Die DNF Methode ist ja sehr speziell, die Aussagen müssen eine spezielle Form haben. Mit welcher Methode kann man auch willkürlich spezifizierte Formeln behandeln.?** [Ja, die sogenannte algebraische Methode]
    a. **Was passiert bei dieser mit Quantoren?** [Allquantoren werden quasi ausmultipliziert, also alle zutreffenden betreffenden Elemente werden einzeln hingeschrieben mit „und“ verknüpft und der Allquantor damit überflüssig, danach kann man die Uniformitäts- und die Regularitätshypothes anwenden| Bei Existenzquantoren, bei denen die Aussage nur auf ein Element zutrifft, kann man

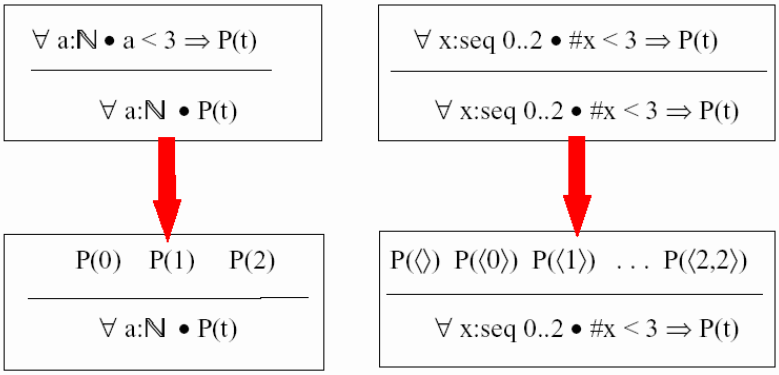
dieses auch direkt hinschreiben, ansonsten „ausmultiplizieren“ d.h. die einzelnen betreffenden Elemente per „oder“ verknüpfen.

b. **Erklären Sie die algebraische Methode an einem Beispiel!** [Insertion Sort: Man benötigt die komplette Spezifikation:

$$\begin{array}{l} perm : seq\ \mathbb{Z} \leftrightarrow seq\ \mathbb{Z} \\ asc \quad : \mathbb{P}(seq\ \mathbb{Z}) \\ \hline \forall\ x, y: seq\ \mathbb{Z} \bullet perm(x, y) = \exists\ f : 1..\#x \rightarrowtail 1..\#x \bullet \\ \qquad\qquad \forall i:1..\#x \bullet x(i) = y(f\,(i)) \\ \forall\ x: seq\ \mathbb{Z} \qquad \bullet asc(x) = \forall\ i:1..\#x\text{-}1 \bullet x(i) \le x(i+1) \end{array}$$

$$\begin{array}{l} insort: \mathbb{Z} \times seq\ \mathbb{Z} \rightarrow \quad seq\ \mathbb{Z} \\ \hline \forall\ a:\mathbb{Z}, x:seq\ \mathbb{Z} \bullet asc\ x \Rightarrow \textbf{let}\ \ y = insort(a, x) \\ \qquad\qquad\qquad\qquad \bullet \quad asc(y) \wedge perm(\langle a \rangle \frown x, y) \end{array}$$

. Zunächst: Einsetzen in Regularitätshypothes(Hypothese wird explizit) und Eliminieren der Quantoren:



Hieraus werden die]Eingabewerte für unsere Testmuster begrenzt. Eingegeben in die Spezifikation erhalten wir unser gewünschtes Test-Set

($H'(x,y,z) = asc(insort(x, y)) \wedge perm(\langle x \rangle \frown y, z)$)]

c. **Was für ein Ergebnis liefern uns nun die generierten Test-Cases?** [Man kann damit feststellen ob das Programm funktioniert oder eben nicht falls ein Gegenbeispiel gefunden wird (Semi-Entscheidbar), aber die Generierung von Test Cases ist sehr rechenaufwendig] (???)

d. **Was können Sie noch über die Algebraische Methode sagen?** [Die Form der Regularitätshypothese zeigt, dass das Testen sich einer Induktion/Verifikation annähert| Man benötigt eine Heuristik zum Festlegen von n| Heuristik zum Bestimmen von |t| | Die Technik ist sehr gut in Verbindung mit Z]