Dagstuhl-Workshop 09153:

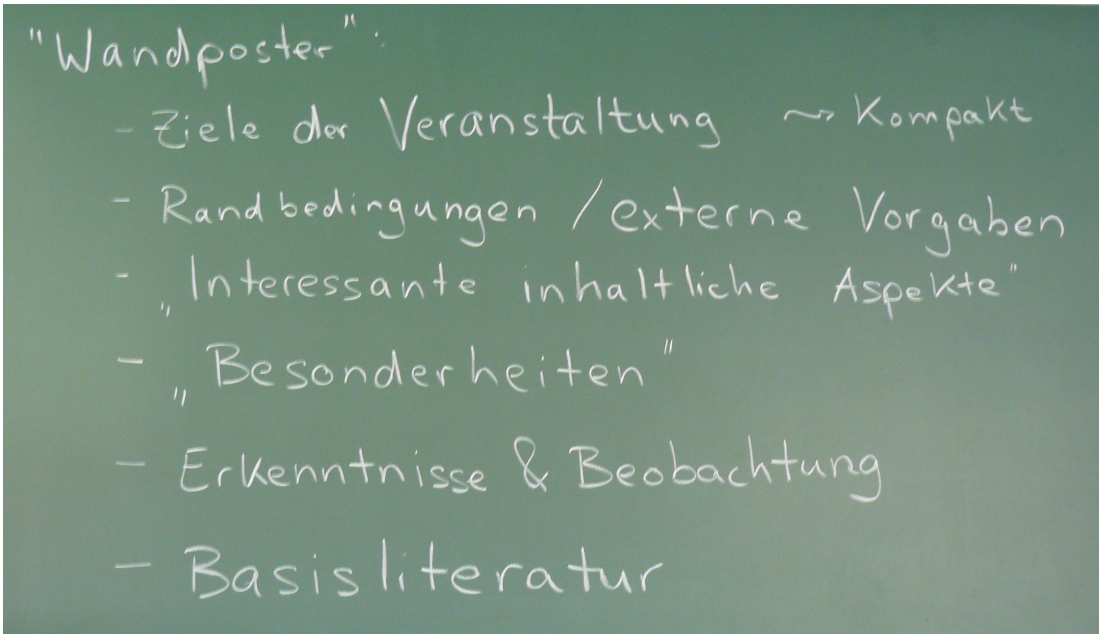
The Intro Programming Course

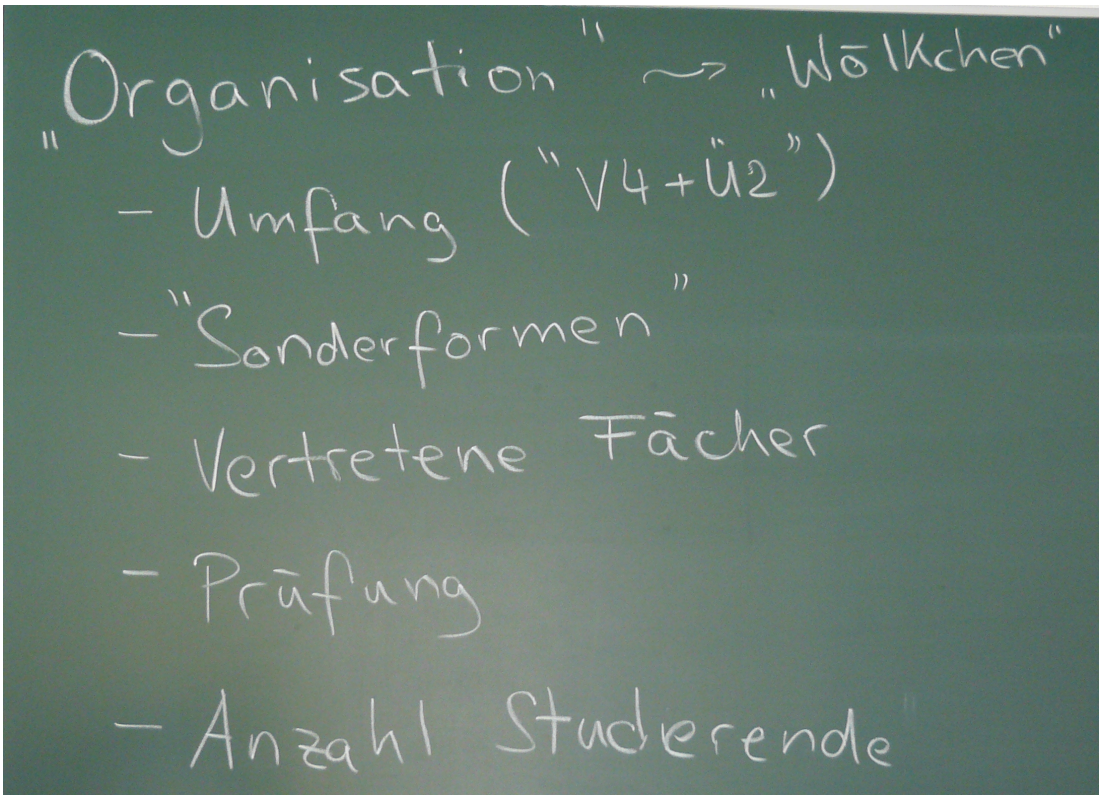
Workshop-Dokumentation

Dieses Dokument ist eine Kurzdokumentation des Workshops für die Teilnehmer, basierend auf den Präsentationen und Visualisierungen, die im Workshop produziert wurden, zusammen mit einigen Notizen und Anmerkungen. Sie dient im wesentlichen als Gedächtnisstütze, nicht aber als vollständiges Protokoll. Begleitend befinden sich noch Foliensätze und Links auf der (nicht-öffentlichen) Seite:

`http://www.deinprogramm.de/dagstuhl-09153/dokumentation/`

1 Vertretene Lehrveranstaltungen




„Organisation" ⇝ „Wölkchen"
- Umfang ("V4+Ü2")
- "Sonderformen"
- Vertretene Fächer
- Prüfung
- Anzahl Studierende

Einführung Nebenfach (RWTH)

Ziele
- Systematisches Progs.
- "Engineering", zum Laufenbringen, Testen
  - ↪ INF. - Arbeitsweisen kennen

Randbed.
- Versch. Einbettung ⇒ Prüfung / LN
- Zeitl. Raster nicht änderbar

Inhalt
- Object first, Java (Ausschnitt)
- Modellierung, Abstraktion
- Didakt. Herausforder.: Aktivität

Besonderheiten
- stark heterogen → Motivation & Vorkenntn.
- Förderstunden, Garantie
- Blended Learning: eTests
  - digit. Übungsbetrieb mit Feedback

Beobachtungen + Erkenntnisse
- Hypothese: - gr. Prozentsatz inaktiv (schreibt ab)
  - Arbeit nicht kontinuierlich
- Maßnahmen:
  - zeitl. eng verzahnter Ü6.-Rhythmus
  - Lernen aus Fehlern, keine Punkte
  - eTest - Fenster

(! Steigerung Aktivität)

Literatur
Java: Head First

Legamaster

Röpling, TU Darmstadt

Ziel: Grundlegendes Verständnis für strukturierte systematische Modellierung & Implementierung

Randbd.: - Teilnehmer aus „fast allen" Studiengängen (tw. nur Einzelpersonen)
- Extreme Heterogenität bei Vorerfahrung Computer / Programmierung
- tw. " Diskrepanz Selbsteinschätzung / unser Eindruck

Inhalt: - systematische Einführung Grundkonzepte Programmierung
~> HtDP „Teaching Languages"
- OO-Programmierung mit Java nach ~6 Wochen
- Starke Betonung Bedeutung Tests [(check-expect;) dann JUnit]

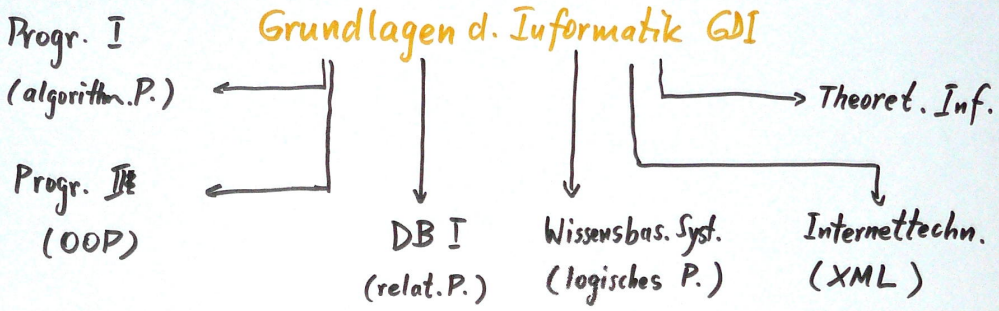
Besonderes: - „(Fast) alles (auch) online" in Portal (inkl. Abgabe / Bew. HÜ, Foren, Aufzeichnungen, ...)
- Blockprojekt ~10d, 4 Wochen Zeit zum "Ein lesen"
- Parallelveranstaltung mit gleichem Inhalt Englisch

Erkenntnisse: - "Viel Stress, viel Zeit, sehr viel gelernt"
- Viel Spaß für Dozent + Studierende
- ↳ sehr hohe Motivation, sehr gute Ergebnisse
- Explizites Lob per EMail auch von „Durchfallern" (~> „selbst schuld")
- Sehr hohe Präsenz Dozent => sehr hohe Präsenz Studierende
=> Sehr hohe Unterstützung "24/7" (o.i.v.) => Motivation & Lernwille
- Skepsis bei „Scheme", aber: „Dozent wird wissen, wieso"

Basislit.: HtDP
SICP
[Thinking in Java]
+ ~~Skript~~ Folien der letzten ~10a "skpwise refined"

Legamaster

HTW Dresden Informatik



Grundlagen der Informatik

Hauptziel: Vermittlung der wichtigsten Programmierparadigmen als Grundlage für die darauf aufbauenden Lehrveranstaltungen (s. oben)

- Verstehen der Paradigmen (Modelle)
- Demonstration (Beispiele)
- keine Fertigkeiten

Besonderheiten:

- Herausheben des deskriptiven Ansatzes
  - OOP -> Abstrakter Datentyp
  - relat. P. -> relationale Algebra
  - logisches P. -> HORN-Klauseln u. Resolution
  - XML -> Metasprachen
- Gegensatz zum algorithm. Paradigma

Programmierung (Uni DUE Duisburg Essen) [Hauptfach]

Ziele: K. & A. strukturierter, objektorientierter Prog.
Nutzung von Standardklassen (Collections) + Bibliotheken
Ausnahme- und Ereignisbehandlung

Randbed: Java Vorbereitung auf Algorithmen & Datenstrukt. vorl. (3.Sem)
Software (2.Sem)

„Projektorientierte Aktivitäten": Dokumentation
Testen
Auslieferung

Analyse der Veranstaltung im Kontext des online-Angebots
Studierendenaktivität (postings etc.) vs. Klausur / Testatpunkte
(fortges. in SW-Vorlesung)
↑ positiv korreliert bei hoher Aktivität

Lit: Empfehlung als Ergänzung Java ist auch eine Insel

Programmiertechnik I (KU Ei) [Nebenfach]

Kenntnis und Anwendungen der Grundlagen der [objektorientierten] Programmierung
Systematik für Strukturierung/Entwurf von Systemen

Randbed: Java Koordination mit Einführung Info 1-3

Objektorientierte und Abstraktionskonzepte
ereignisbasierte Programmierung ⇝ GUI als Anwendungsbeispi
Umgang mit Bibliotheken / vorgef. Komponenten

Konzepte vor Spracheigenheiten
Revision von Programmcode (⇝ Refactoring)

0% Powerpoint ad hoc Folien mit integrierter Info aus Entwicklungsumgebung

100% passing rate ☺ [niedrige Zahlen]

Informatik I

Universität Freiburg

Ziele:
- Systematisches Programmieren
- theoretische Grundlagen (Induktion, Mengen, Relationen, Lambda-Kalk
- Einblick in funktionales/imperatives/OO-Paradigma

Randbedingungen:
- Algorithmenvorlesung in Java (2. Sem.)
- einzige "Programmier"vorlesung im Curriculu
- Kollegenwunsch: Java 1. Sem., C-Kenntnisse im 5. Sem.
- Erweiterung auf 4 VL + 2 ÜB + 2 BP

Inhaltl. Aspekte:
- funktionale Programmierung
- Konstruktionsanleitung
- Testen + Dokumentieren

Zusammenarbeit mit Tübinge
⇒

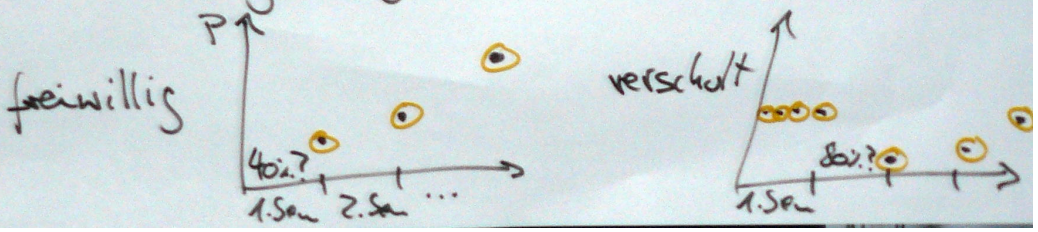
Beobachtungen:
- stark differenziertes Anfangslevel (Probleme im Umgang mit Computer, Tippen, ...)
- "zu viele falsche" Vorkenntnisse stören
- strenger organisatorischer Rahmen notwendig

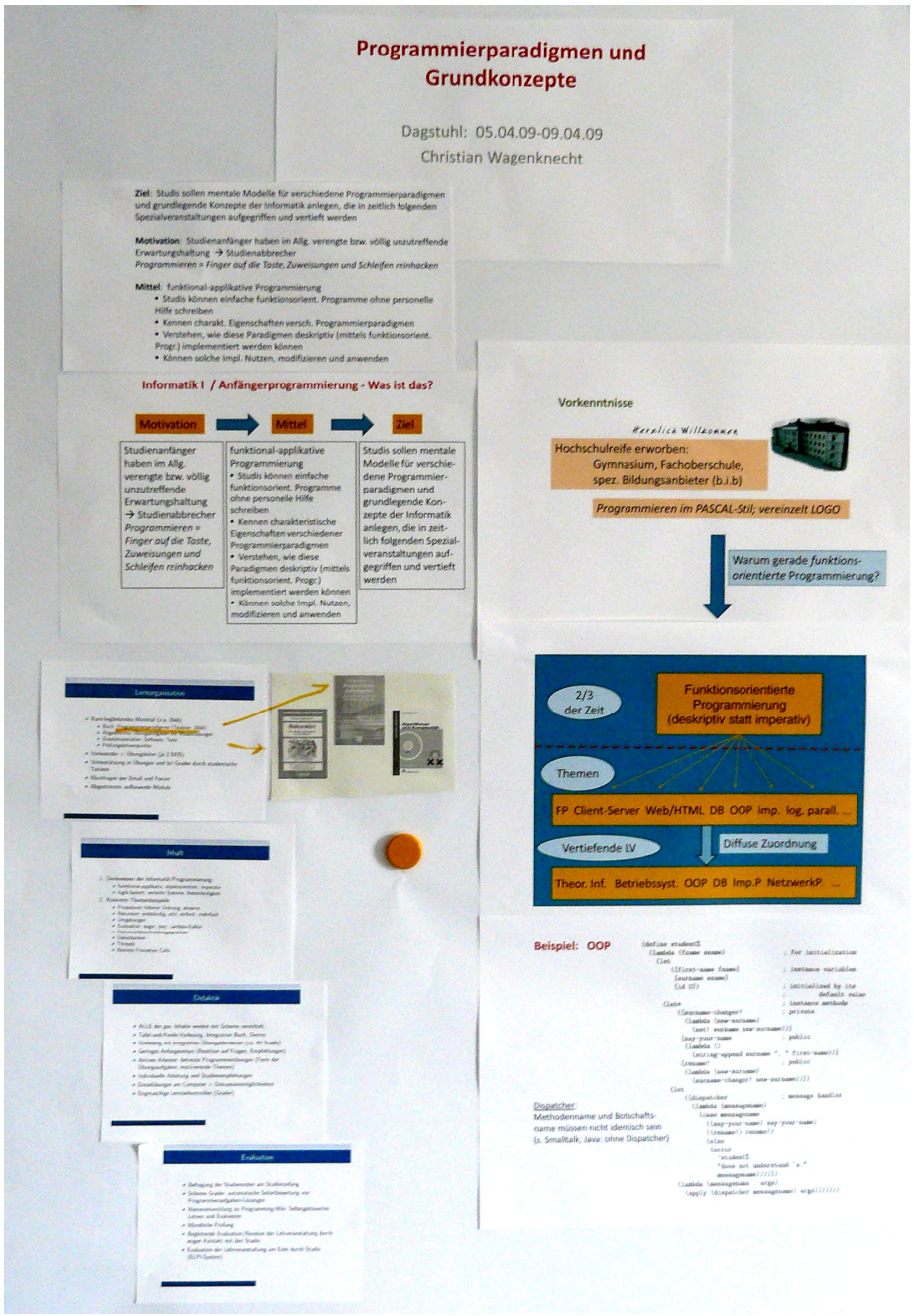
Hochschule Zittau/Görlitz (FH)
„Einführung in die Programmierung"

Ziele:
- Handlungskompetenz in Programmierung
- Kommunikationsfähigkeit, Teamarbeit
- Spaß an Informatik

Randbed.:
- Kurs läuft Hand in Hand mit „Programmierparadigmen"
  → Abstimmung d. Inhalte
- Erwartet wird Java-Kompetenz ab dem 3. Sem.

Erkenntnisse:
1. Beteiligung an freiwilligen Übungen korreliert mit Leistung
2. nicht-freiwillige Übungen sind langfristig schädlich





PDF-Version auf der Dokumentations-Webseite:

`http://www.deinprogramm.de/dagstuhl-09153/dokumentation/programmierparadigmen-wagenknecht.pdf`

PR1 + PR2 (HAW)

Ziele

- Progr. als Modellierung
- Konstruktion kleiner SW-Maschinen
- mentales Modell einer abstrakten OO-Maschine

Randbedingungen

- Erlernen einer praxisrelevanten PS
- Gemeinsame Sprache (Inform./Techn. Inf.)
  ⟶ Java in 2. Sem.

Inh. Aspekte:

- Abstraktionstechniken
- Entwurfstechniken, Korrektheit

Besonderheiten:

- 1. Sem Ruby (2/3), 2. Sem. Java (1/3)

Erkenntnisse

- Verstehen fremder Programme ist wichtig
- Divergenz von Leistungsgruppen 


HTWK LEIPZIG
PROGRAMMIEREN / JAVA (1. SEM.)
ZIELE:
• Paradigmen
• einfaches Problem-lösen
• Einführung in Java
RANDBED.:
• extrem wenig Zeit
• „nur ein Teil"
• parallel: C
INHALT:
- „Pseudo"-Funktional
- Imperativ
- OO
mit Hamster-Java
- Schnittstellen
- IO
- Exceptions
Java
BESONDERHEITEN:
- Präsenzübung
- 3 Programmierbelege
- anonyme Probeklausur
Funktionale Tests, Plagiate, Programmierrichtlinien, ...
- ggf. „Schein"-Projekt
ERKENNTNISSE:
- „wiederschwelligen" Einstieg positiv
- subjektiv: bessere Prog.kenntnisse als bei „Standard-Java-Lehre"
LITERATUR:
zu viele „Steinbrüche"
Legamaster
We make communication visible
www.legamaster.com

CS 1+2 at NEU

1) Ziel: design of programs, computational model
   CS 2: basic data struct libs

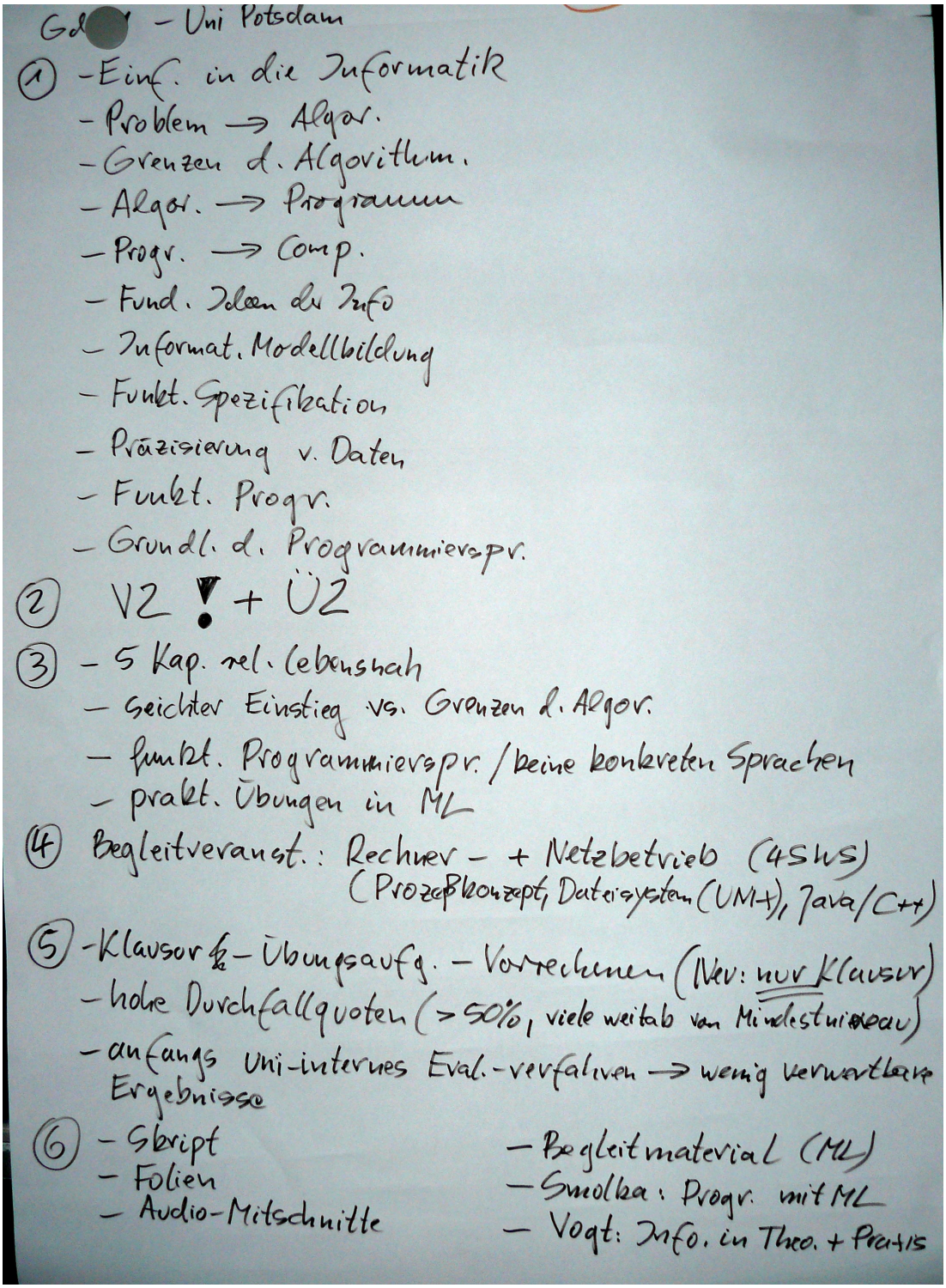
2) CS 2: Brücke zu Java
   (downstream courses)

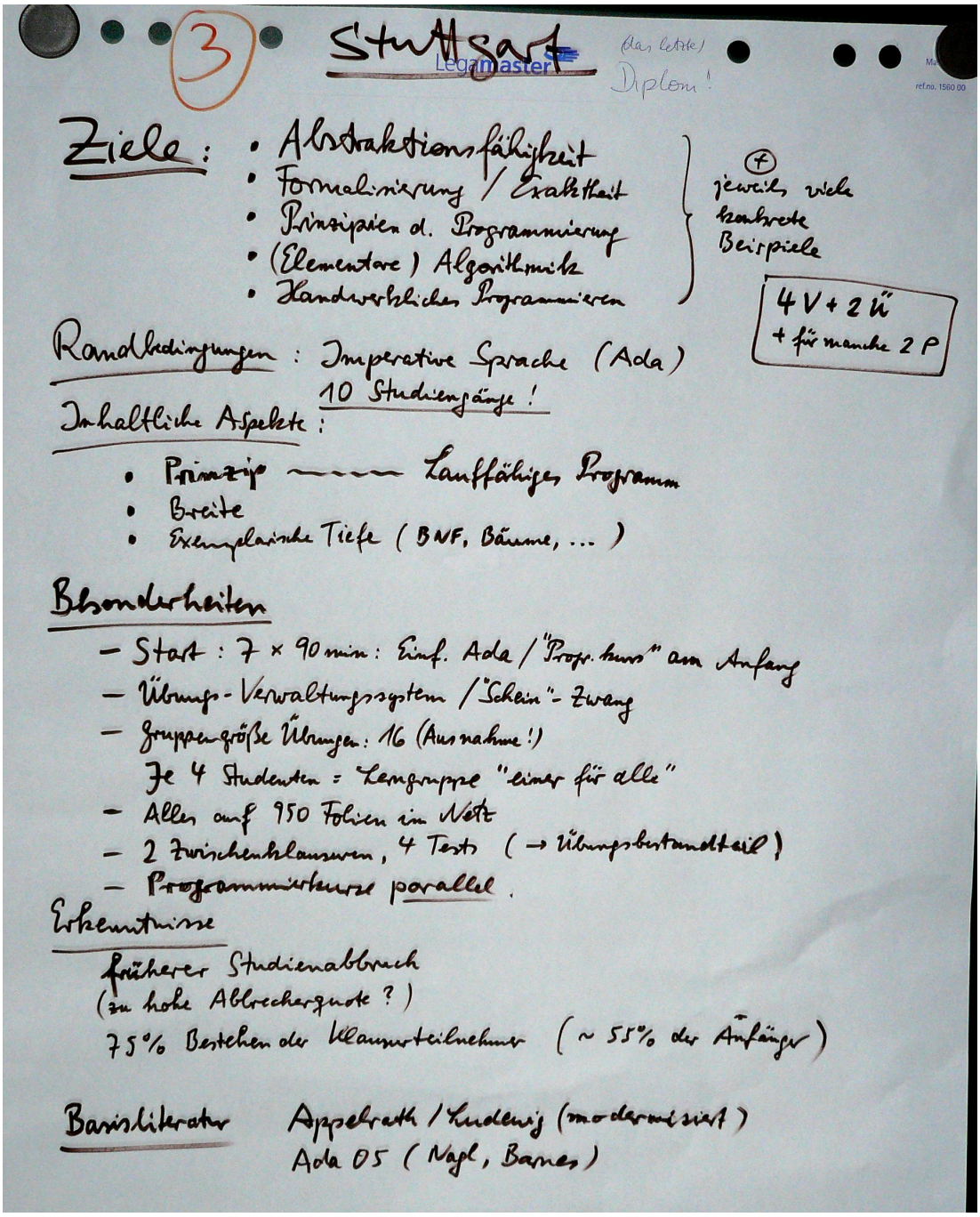
3) Design Recipe, teaching languages, pedag. IDE
   reactive, distributed context

4) CS 1 widely populated by non-majors

5) — our course has improved instruction at
   large (Toronto), medium-size (NEU), small (Lesser)
   Universities

6) How to Design Programs (2e) MIT ?
   How to Design Classes to appear


Gö[redacted] – Uni Potsdam
① – Einf. in die Informatik
– Problem → Algor.
– Grenzen d. Algorithm.
– Algor. → Programm
– Progr. → Comp.
– Fund. Ideen der Info
– Informat. Modellbildung
– Funkt. Spezifikation
– Präzisierung v. Daten
– Funkt. Progr.
– Grundl. d. Programmierspr.
② V2 ! + Ü2
③ – 5 Kap. rel. lebensnah
– leichter Einstieg vs. Grenzen d. Algor.
– funkt. Programmierspr. / keine konkreten Sprachen
– prakt. Übungen in ML
④ Begleitveranst.: Rechner – + Netzbetrieb (4SWS)
(Prozeßkonzept, Dateisystem (UM+), Java/C++)
⑤ – Klausur & – Übungsaufg. – Vorrechnen (Nev: nur Klausur)
– hohe Durchfallquoten (> 50%, viele weitab von Mindestniveau)
– anfangs Uni-internes Eval.-verfahren → wenig verwertbare Ergebnisse
⑥ – Skript
– Folien
– Audio-Mitschnitte
– Begleitmaterial (ML)
– Smolka: Progr. mit ML
– Vogt: Info. in Theo. + Praxis


3
Stuttgart
(das letzte)
Diplom!
Ziele:
• Abstraktionsfähigkeit
• Formalisierung / Exaktheit
• Prinzipien d. Programmierung
• (Elementare) Algorithmik
• Handwerkliches Programmieren
+ jeweils viele konkrete Beispiele
4 V + 2 Ü
+ für manche 2 P
Randbedingungen: Imperative Sprache (Ada)
10 Studiengänge!
Inhaltliche Aspekte:
• Prinzip ——— Lauffähiges Programm
• Breite
• Exemplarische Tiefe (BNF, Bäume, ...)
Besonderheiten
– Start: 7 × 90 min: Einf. Ada / "Progr. kurs" am Anfang
– Übungs-Verwaltungssystem / "Schein"-Zwang
– Gruppengröße Übungen: 16 (Ausnahme!)
Je 4 Studenten = Lerngruppe "einer für alle"
– Alles auf 950 Folien im Netz
– 2 Zwischenklausuren, 4 Tests (→ Übungsbestandteil)
– Programmierkurse parallel.
Erkenntnisse
früherer Studienabbruch
(zu hohe Ablehnergnote?)
75% Bestehen der Klausurteilnehmer (~ 55% der Anfänger)
Basisliteratur Appelrath / Ludewig (modernisiert)
Ada 05 (Nagl, Barnes)

Informatik I

Tübingen

Ziele:

- Einführung in die Programmierung (70%)
- systematisches Vorgehen
- theoretische Grundlagen (30%)
- hohe Lernerfolgsquote

externe Vorgaben:

- ignorieren wir

inhaltliche Aspekte:

- HtDP
  - KA
  - Sprachen
  - IDE

Besonderheiten:

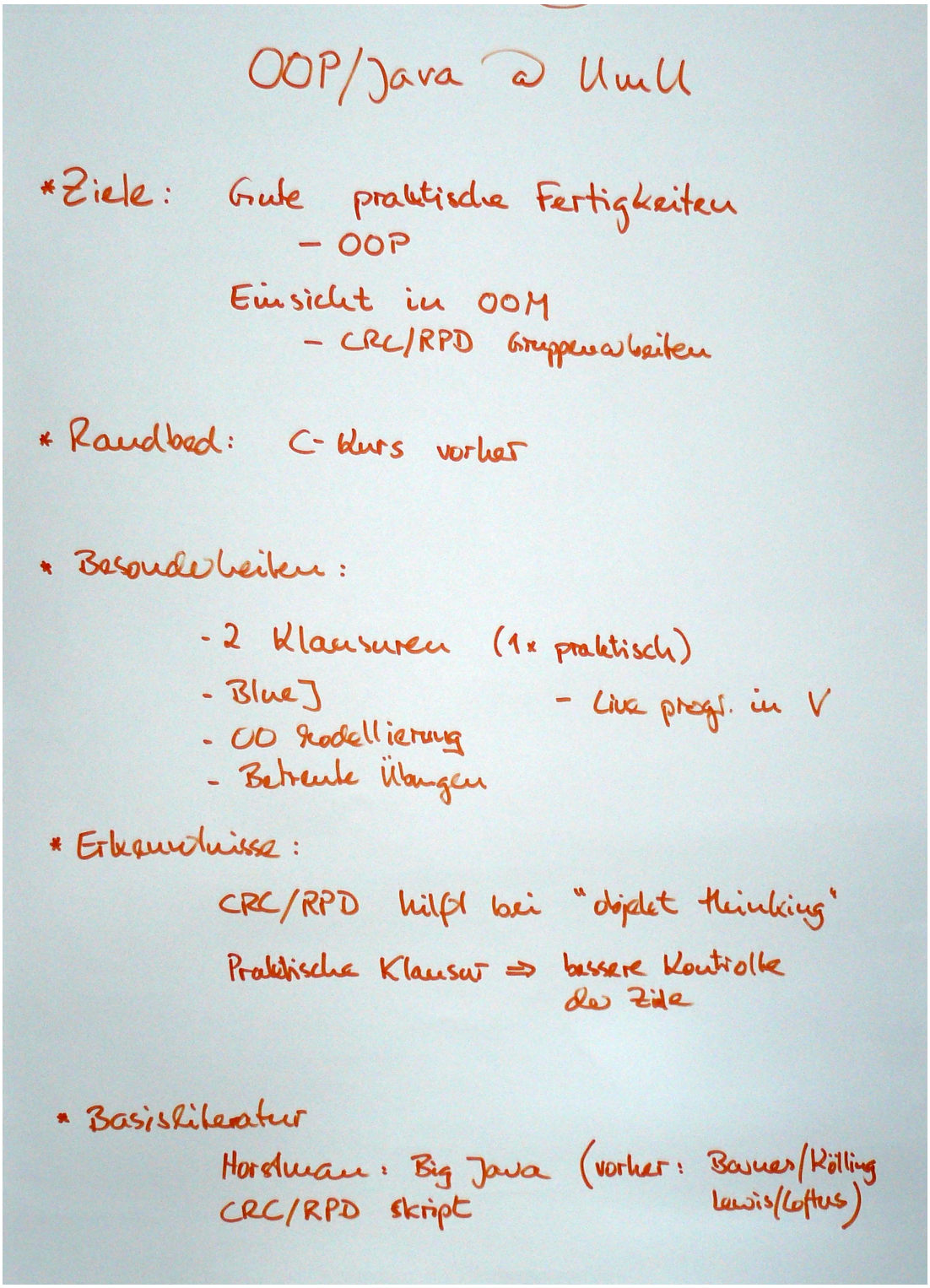
- Präsenzübungen
- Testate
- Gruppentestate
- kleine Übungsgruppen (~ 8-10 Tln.)
- aktive Plagiatbekämpfung
- Zusammenarbeit mit Freiburg

Erkenntnisse und Beobachtungen:

- Korrelationen:
  - PÜ ↔ Klausur
  - ÜB ↔ Klausur
  - Testate ↔ Klausur
- Studenten wollen gefordert werden
- 'strenge' Führung

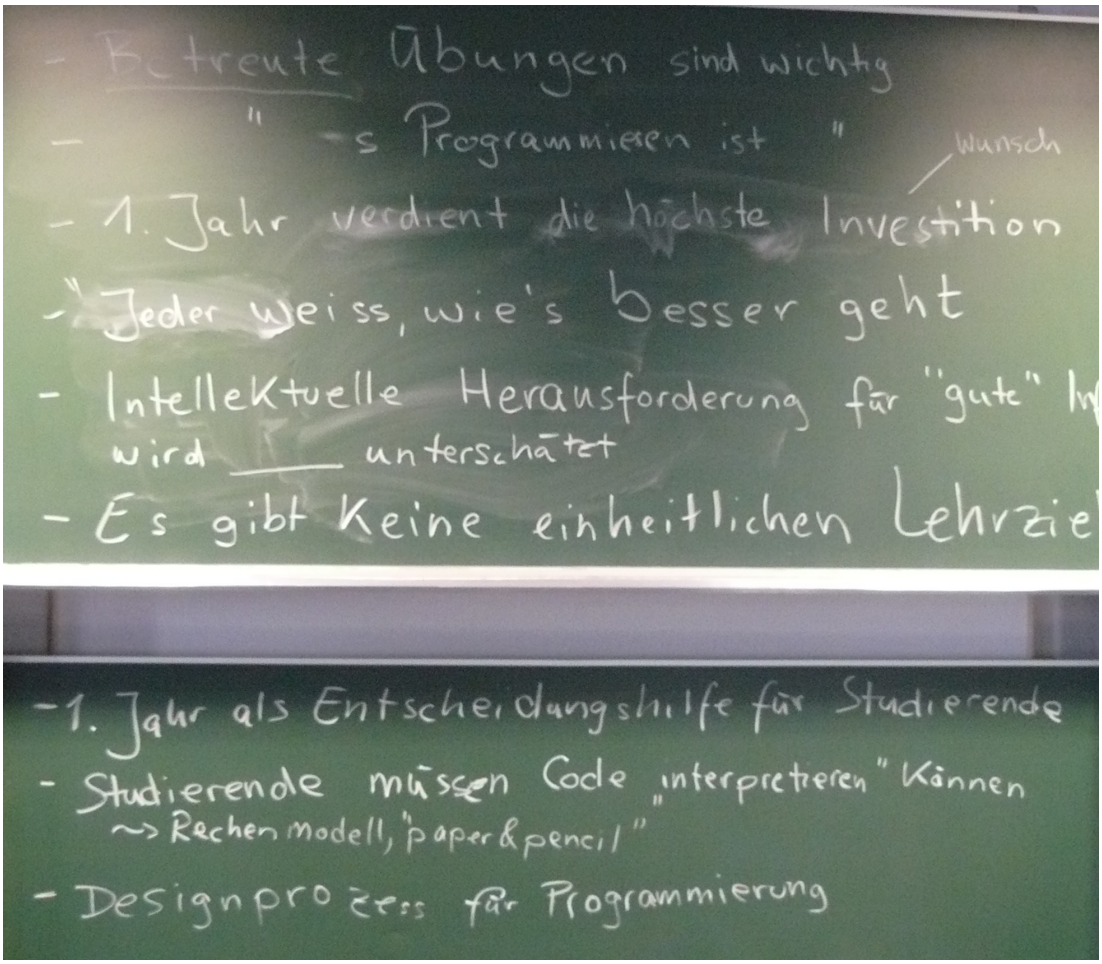
Literatur:

- DMdA
- Forum

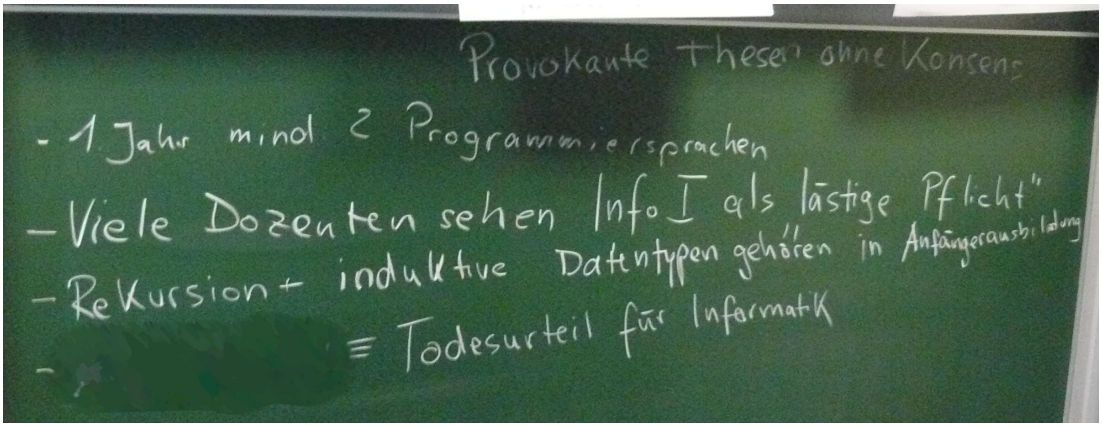

OOP/Java @ UniU
*Ziele: Gute praktische Fertigkeiten
- OOP
Einsicht in OOM
- CRC/RPD Gruppenarbeiten
* Randbed: C-Kurs vorher
* Besonderheiten:
- 2 Klausuren (1x praktisch)
- BlueJ
- OO Modellierung
- Betreute Übungen
- Live progr. in V
* Erkenntnisse:
CRC/RPD hilft bei "disjunct thinking"
Praktische Klausur ⇒ bessere Kontrolle des Zils
* Basisliteratur
Horstmann: Big Java (vorher: Barnes/Kölling
CRC/RPD skript Lewis/Loftus)

2 Konsens und Provokation

In dieser Schnellsitzung warfen die Teilnehmer Thesen ein, die sie geeignet für einen Konsens *ohne Diskussion* hielten:
Die folgenden Thesen waren schnell konsensfähig:

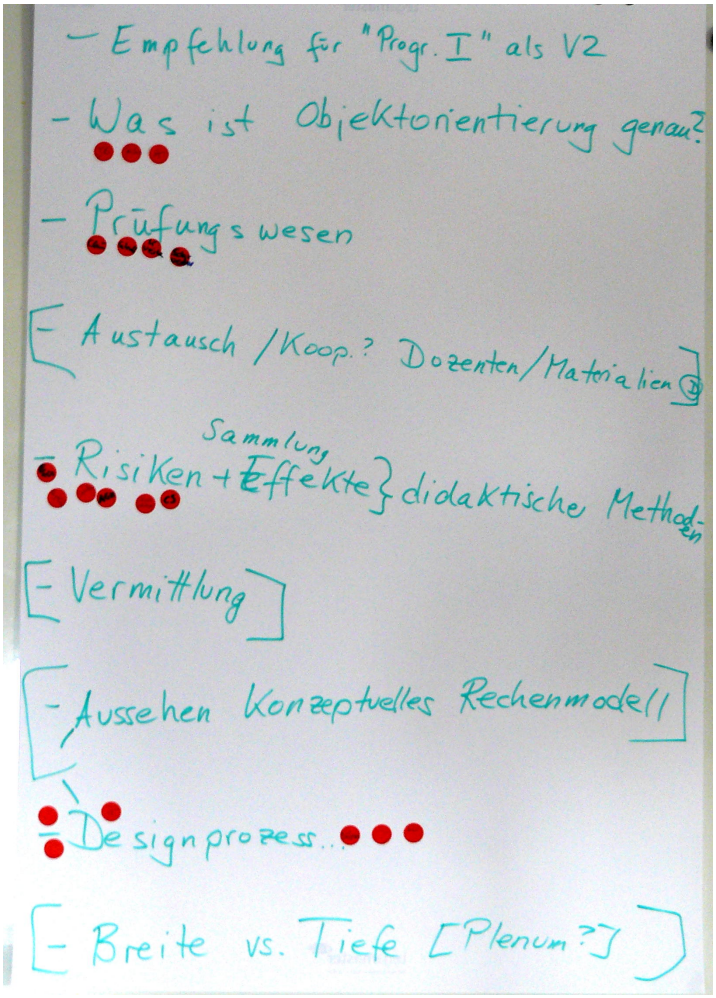


Bei den folgenden Thesen gab es keine schnelle Einigung:



3 Themen für Gruppenarbeit

Hier die vorgeschlagenen Themen für intensive Gruppenarbeit:



Durch Setzen von Punkten wurden die Themen für die Bearbeitung ausgewählt. Für jedes Thema wurden Kleingruppen gebildet. Vorher wurden jeweils noch jeweils ein „Päckchen“ mit Vorschlägen der Teilnehmer aus den anderen Gruppen geschnürt.

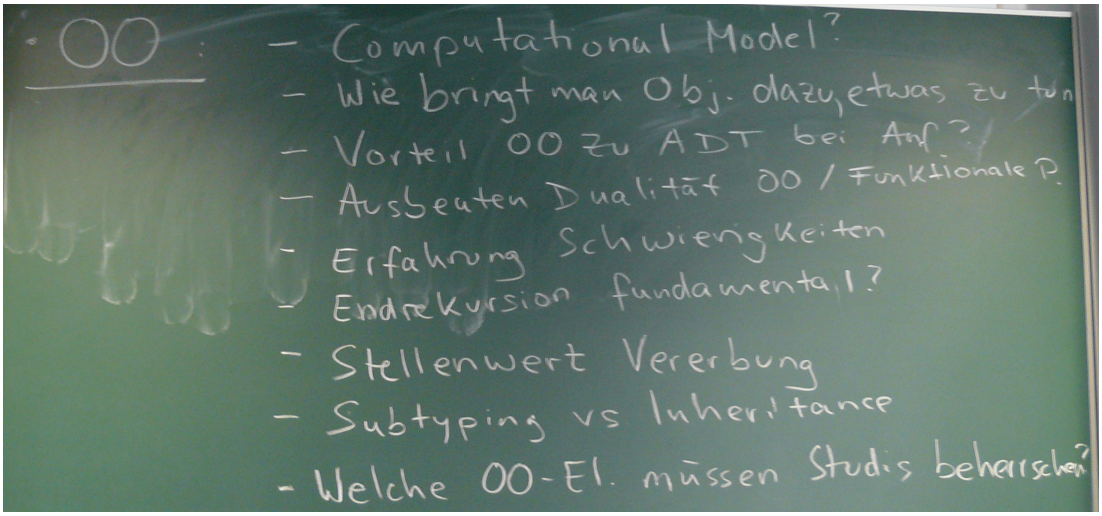
3.1 Was ist objektorientierte Programmierung

Teilnehmer

- Annette Bieniusa
- Jürgen Börstler
- Ralph Großmann

- Peter Thiemann

Päckchen



Folien

... separat auf der Dokumentations-Seite:

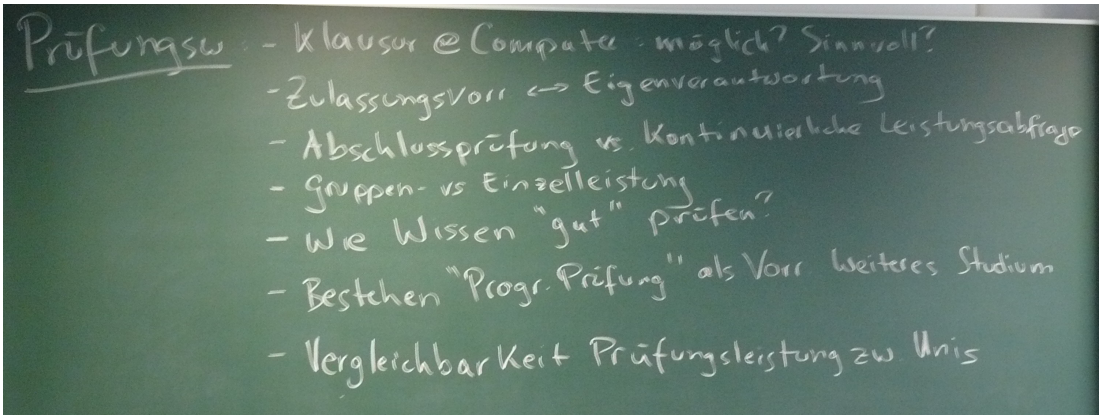
`http://www.deinprogramm.de/dagstuhl-09153/dokumentation/objekte.pdf`

3.2 Prüfungswesen

Teilnehmer

- Volker Claus
- Andreas Schwill
- Ulrik Schroeder
- Karsten Weicker

Päckchen



Folien

... separat auf der Dokumentations-Seite:

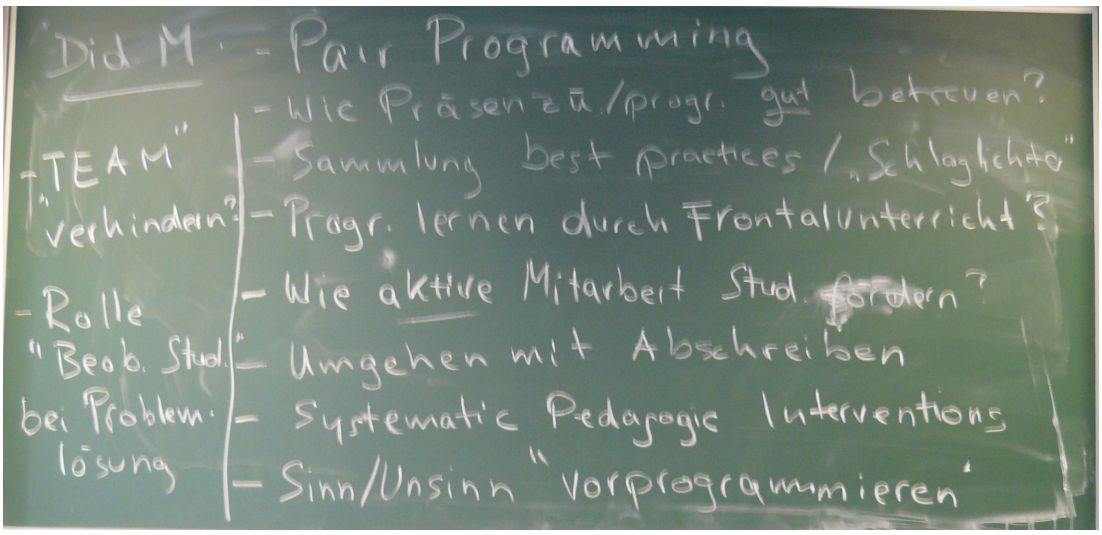
`http://www.deinprogramm.de/dagstuhl-09153/dokumentation/pruefungswesen.pdf`
`http://www.deinprogramm.de/dagstuhl-09153/dokumentation/pruefungswesen-einordnungsschema.pdf`

3.3 Didaktische Methoden

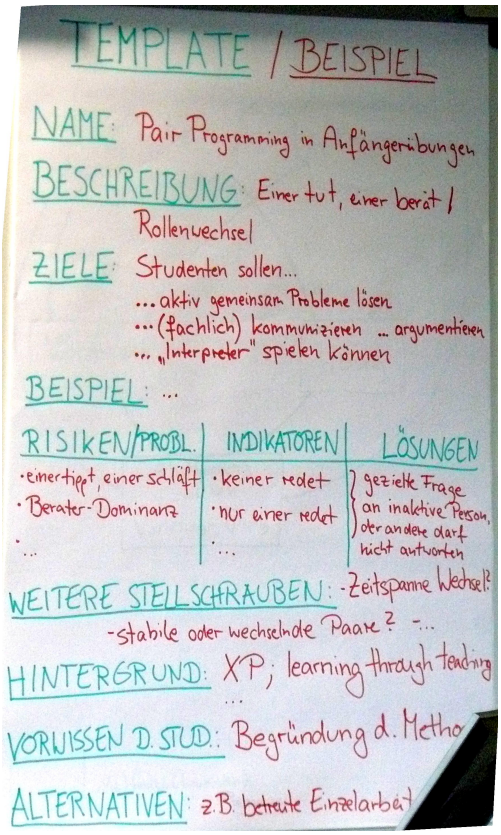
Teilnehmer

- Eva Altenbernd-Giani
- Marcus Crestani
- Andreas Harrer
- Georg Ringwelski
- Nicole Weicker

Päckchen



Vortrag

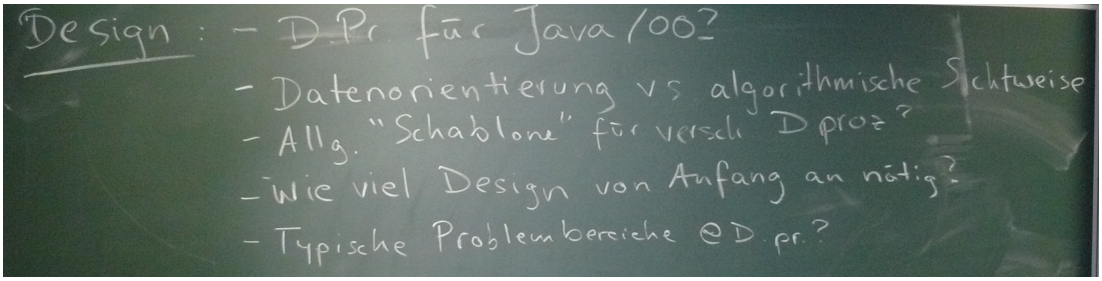


3.4 Designprozess

Teilnehmer

- Michael Böhm
- Matthias Felleisen
- Stefan Wehr
- Birgit Wendholt

Päckchen



Folien

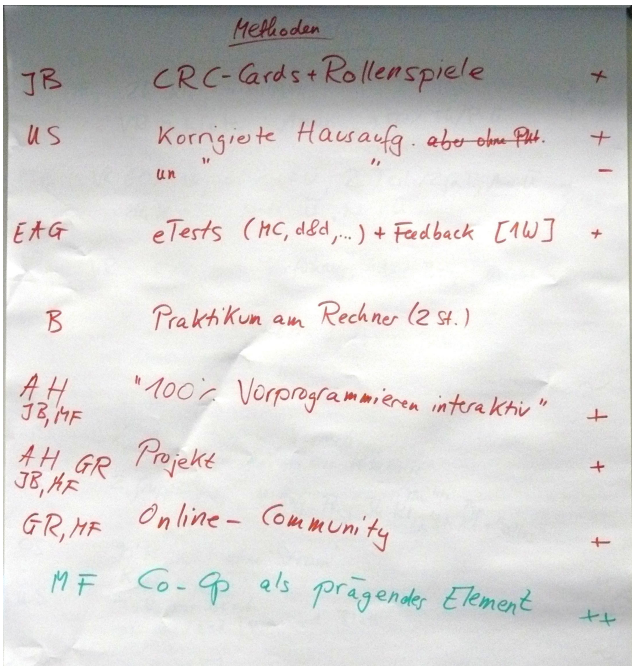
... separat auf der Dokumentations-Webseite:

`http://www.deinprogramm.de/dagstuhl-09153/dokumentation/designprozess.pdf`.

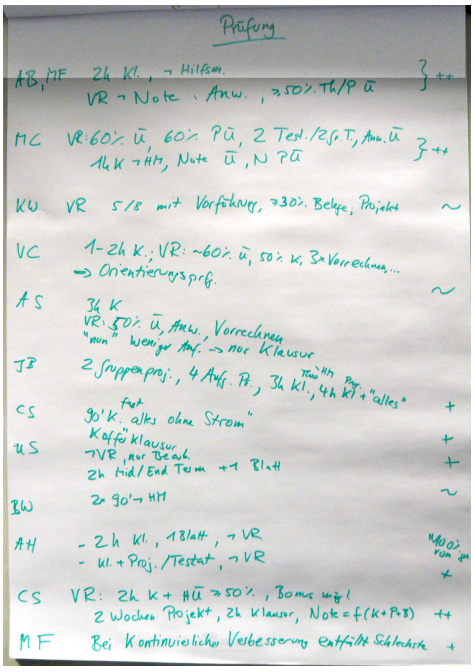
4 Abschlußsitzung Dienstag

4.0.1 Zuruflisten zu Erfahrungen

Zurufliste mit Erfahrungen zu verschiedenen didaktischen Methoden:



Zurufliste mit Erfahrungen zu verschiedenen Methoden im Prüfungswesen:



4.0.2 Erfahrungen mit reinen Hauptfach- / HF+NF-Veranstaltungen

Die vertretenen Lehrveranstaltungen haben unterschiedliche Zusammensetzungen:

- nur Hauptfach

- Hauptfach und Nebenfach
- nur Nebenfach

Generell ist es möglich, dass sich die Kombination von Hauptfach und Nebenfach erschwerend auf die Lehrveranstaltung auswirkt. Dies scheint aber von anderen Begleitumständen abzuhängen; mehrere kombinierte Lehrveranstaltungen berichten, dass es keine besonderen Probleme geht.

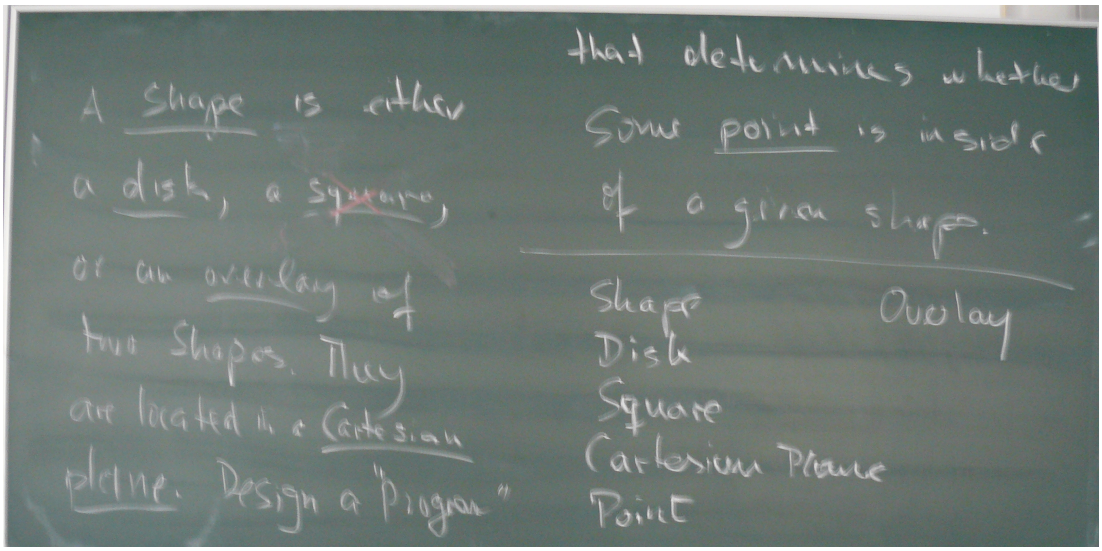
4.0.3 Terminologische Klärung Polymorphie vs. Überladung

Java und andere Programmiersprachen enthalten zwei verschiedene Mechanismen, die gelegentlich unter dem Oberbegriff „Polymorphie“ geführt werden:

- Bei *parametrischer Polymorphie* ist eine Operation unabhängig vom Typ ihrer Argumente, funktioniert also für verschiedene Typen gleichartig.
- Bei *Überladung* werden mehrere Operationen unter einem Namen zusammengefasst. Die tatsächlich angewendete Operation wird dabei nach dem Typ der Argumente ausgewählt. Überladung wird auch als *Ad-hoc-Polymorphie* (zurückgehend auf Strachey) benannt.

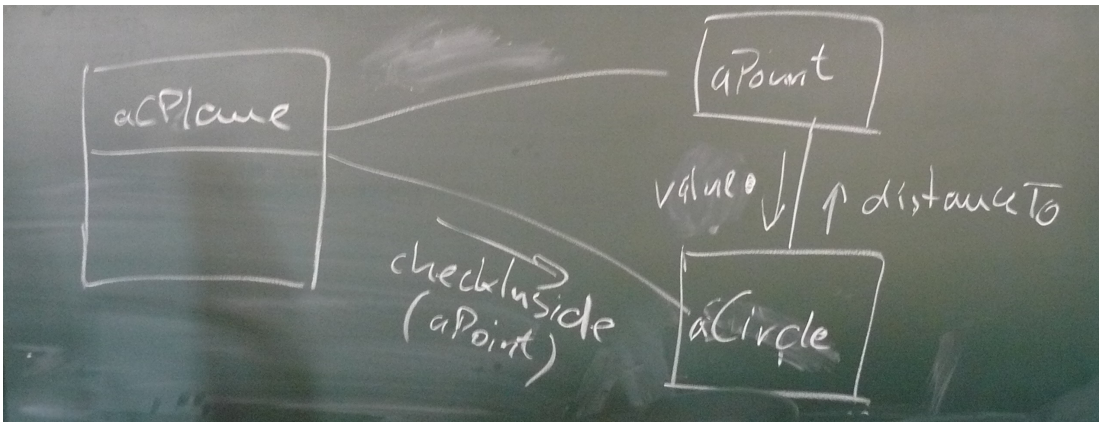
5 Sitzung zum Thema „Designprozess“

Jürgen Börstler, Matthias Felleisen und Michael Sperber berichten von ihrer informellen Sitzung vom Abend vorher: Sie wählten eine Beispielaufgabe aus und vollzogen verschiedene Lösungs-Methodiken nach. Hier eine Kurzzusammenfassung – mehr Informationen in Bälde im Wiki.
Beispielproblem:



Ansatz mit CRC-Karten und Szenarios

Bei diesem Ansatz werden zunächst die Substantive aus der Problemstellung extrahiert (Figur, Kreis, Quadrat, Überlappung, kartesische Ebene, Punkt). Für jedes dieser Substantive wird eine CRC-Karte (der später eine Klasse zugeordnet wird) angelegt mit Kurzbeschreibung, „Responsibilities“ und „Collaborators“ angelegt.
Dann wird im Team ein Rollenspiel durchgeführt, in dem Objekte und mögliche Methodenaufrufe eingezeichnet werden:



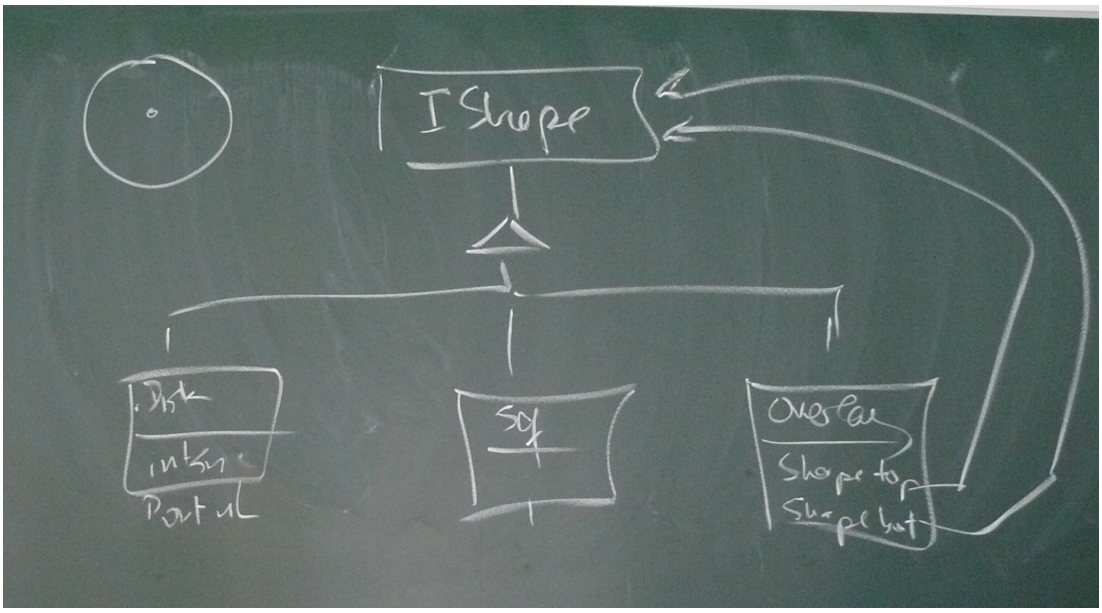
Die CRC-Karten helfen also dabei, die Rollen von Klassen und Objekten im System herauszufindenn und dienen dann als Skizze für den späteren Code, bietet aber keine weitere Hilfestellung bei der Programmierung.

Ansatz aus „How to Design Programs“ / „How to Design Classes“

Bei diesem Ansatz wird zuerst eine systematische Datenanalyse durchgeführt, die folgende Ergebnisse liefert:

- Bei „Figur“ handelt es sich um *gemischte Daten*, d.h. eine Figur kann eine von drei verschiedenen Arten von Figuren sein.
- Bei Quadrat, Kreis und Überlappung liegen jeweils *zusammengesetzte Daten* vor: Ein Quadrat besteht aus Eckpunkt und Größe, ein Kreis aus Mittelpunkt und Radius, die Überlappung aus zwei Figuren.
- Bei Überlappungen liegt eine *Selbstreferenz* vor, da sie jeweils zwei Verweise auf Figuren enthalten.

Daraus entsteht bei der direkt ein Klassendiagramm (nur bei *How to Design Classes*):

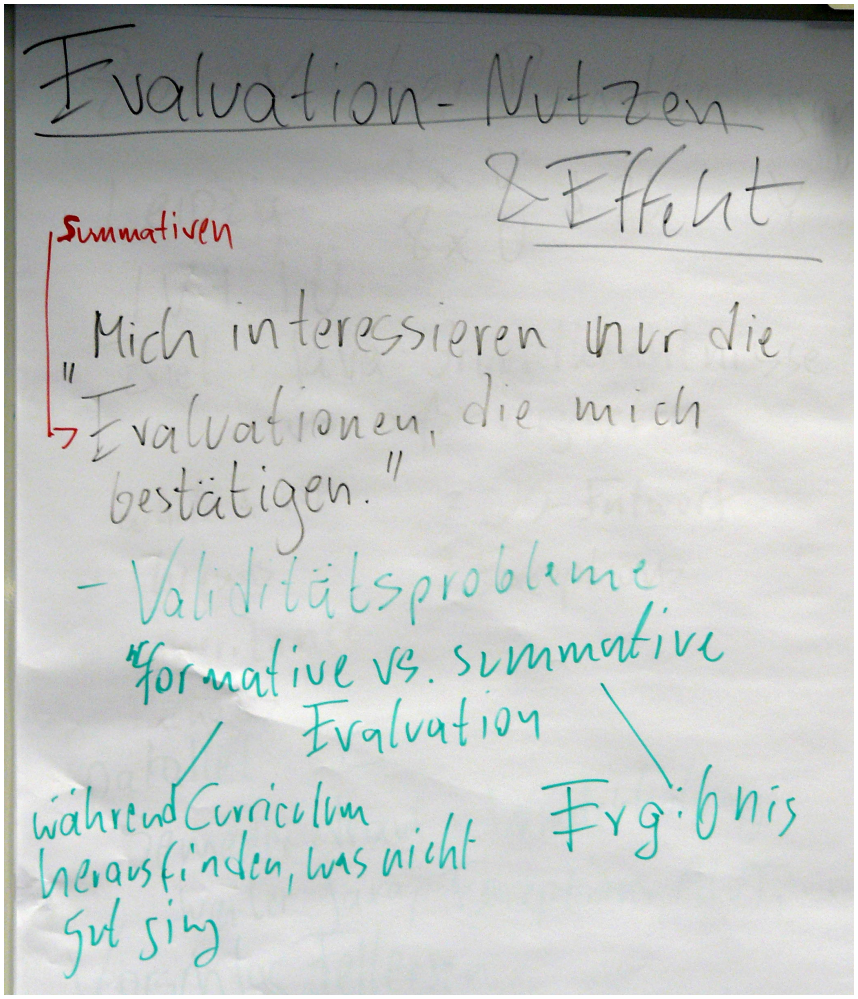


Daraus wiederum entsteht direkt auf semi-mechanische Art und Weise der Code: Die Daten bestimmen Schablonen für die Klassen und Methoden, die dann ausegüllt werden. Es wird klar getrennt zwischen dem Prozess der Programmstrukturierung und dem Einbringen bereichsspezifischen Wissens (Pythagoras etc.). Der Designprozess reicht, im Gegensatz zu traditionellen OO-Design-Methoden bzw. Sofware-Engineering-Prozessen, „bis ganz unten“.

Zwischenfazit

Programmieranfänger scheitern oft dabei, aus einer Aufgabenstellung erfolgreich ein Programm zu schreiben. Die Fähigkeiten und Techniken, die dazu notwendig sind, müssen explizit vermittelt werden, damit diese Programmieranfänger zum Erfolg kommen: Es reicht nicht, über Beispiele zu lehren. Inbesondere genügt es oft nicht, die „Grobstruktur“ im Rahmen des Designprozesses zu entwerfen und die Ausgestaltung durch Code den Anfängern dann selbst zu überlassen.

6 Evaluation – Nutzen und Effekt

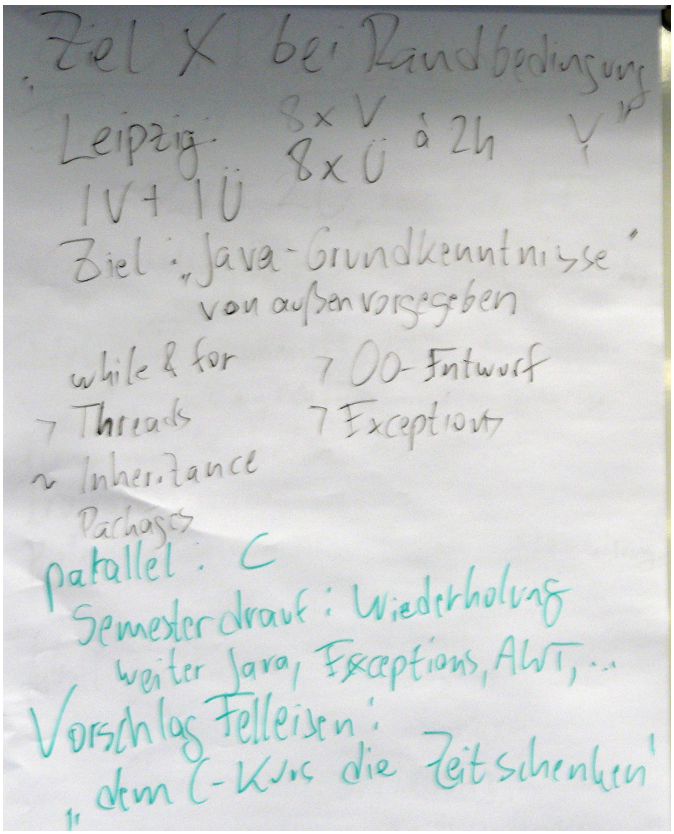


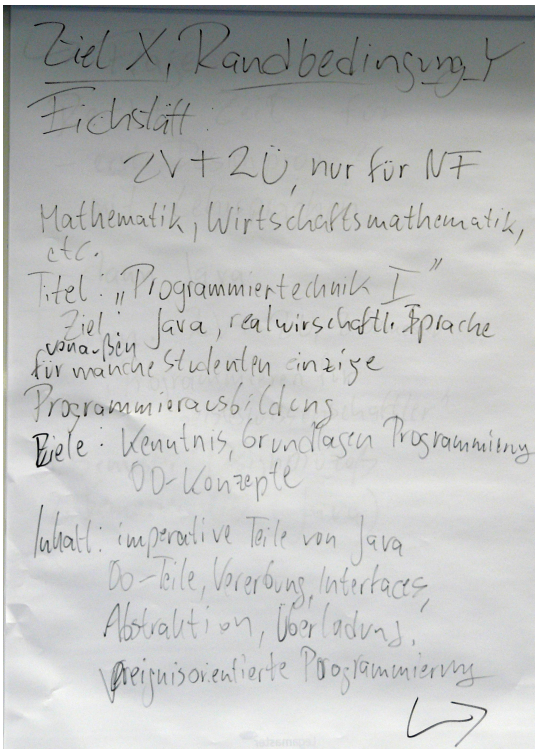
7 Anmerkungen zum ACM/IEEE-Curriculum

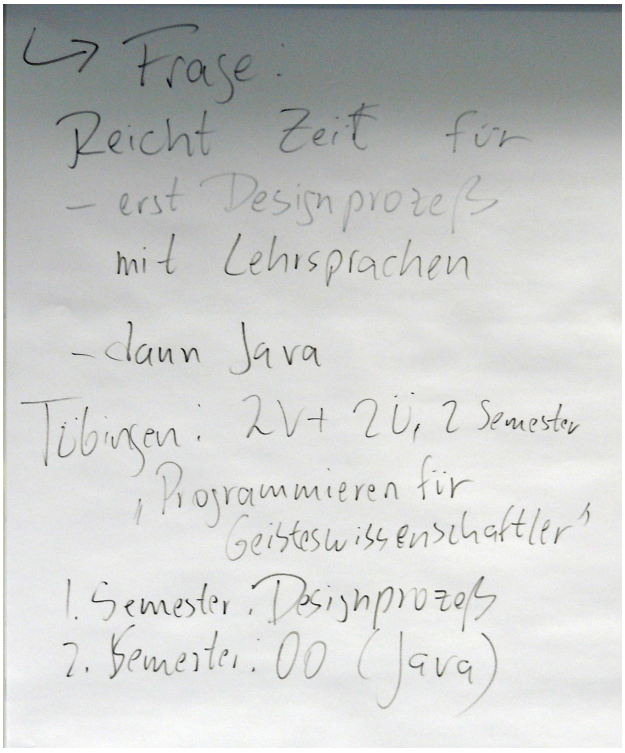
Matthias Felleisen berichtet über die Aktivitäten der SIGPLAN bezüglich der curricularen Empfehlungen von ACM und IEEE zum „undergraduate curriculum“ in Informatik aus dem Jahr 2001. Obwohl fast alle curricularen Empfehlungen ab 1968 eine Empfehlung enthalten hatten, einen Kurs zum Thema „Programmiersprachen“ zu unterrichten, enthielt die 2001-Empfehlung diesen Kurs nicht mehr.
2008 fand ein SIGPLAN-Workshop zum *Programming Languages Curriculum* statt, der die Empfehlung verabschiedete, einen Programmiersprachen-Kurs wieder im Curriculum aufzunehmen. Insbesondere sollte funktionale Programmierung einen Platz im Curriculum finden. Die Empfehlung bekam enthusiastische Unterstützung über die Internet-Kommentare, fand aber keine Zustimmung im

„review committee“.
Mehr Informationen auf dem Wiki.

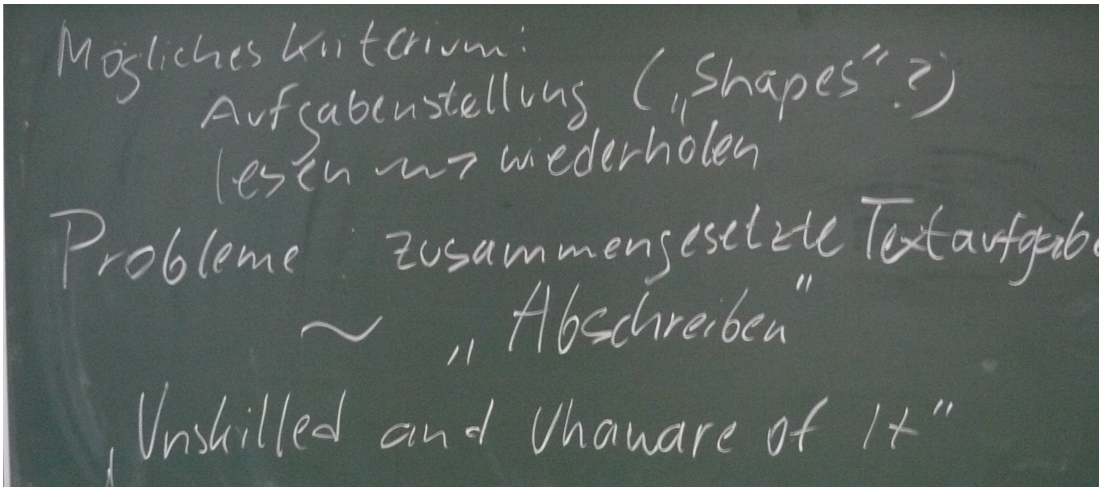
8 Ziele vs. Randbedingungen



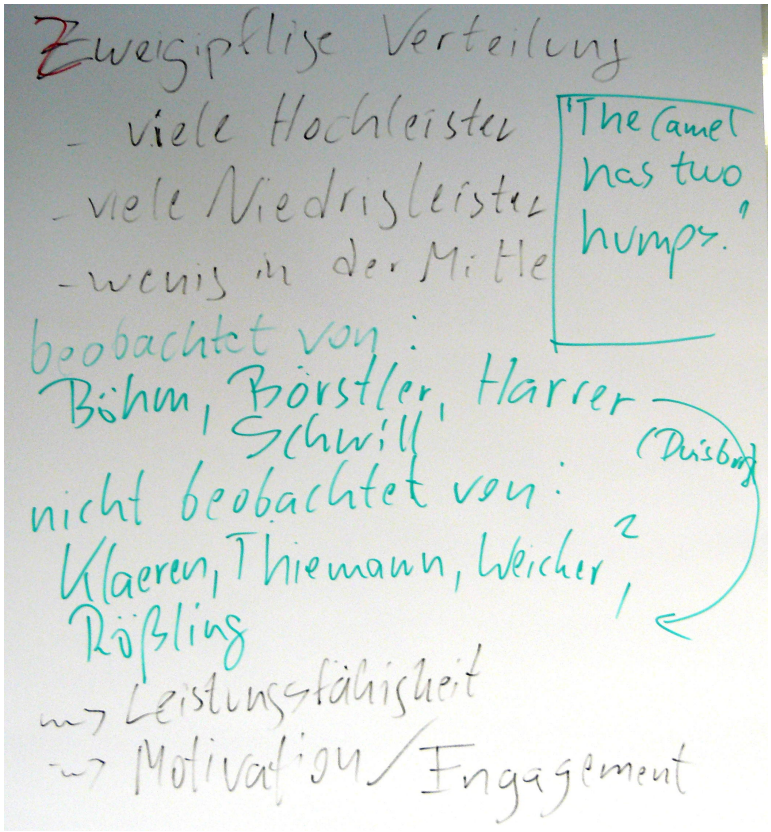

Ziel X, Randbedingung Y
Eichstätt:
2V + 2Ü, nur für NF
Mathematik, Wirtschaftsmathematik, etc.
Titel: „Programmiertechnik I"
Ziel: Java, realwirtschaftl. Sprache
sozusagen für manche Studenten einzige Programmierausbildung
Ziele: Kenntnis, Grundlagen Programmierung
OO-Konzepte
Inhalt: imperative Teile von Java
OO-Teile, Vererbung, Interfaces, Abstraktion, Überladung,
ereignisorientierte Programmierung


Frage:
Reicht Zeit für
- erst Designprozess mit Lehrsprachen
- dann Java
Tübingen: 2V + 2Ü, 2 Semester
„Programmieren für Geisteswissenschaftler"
1. Semester: Designprozess
2. Semester: OO (Java)

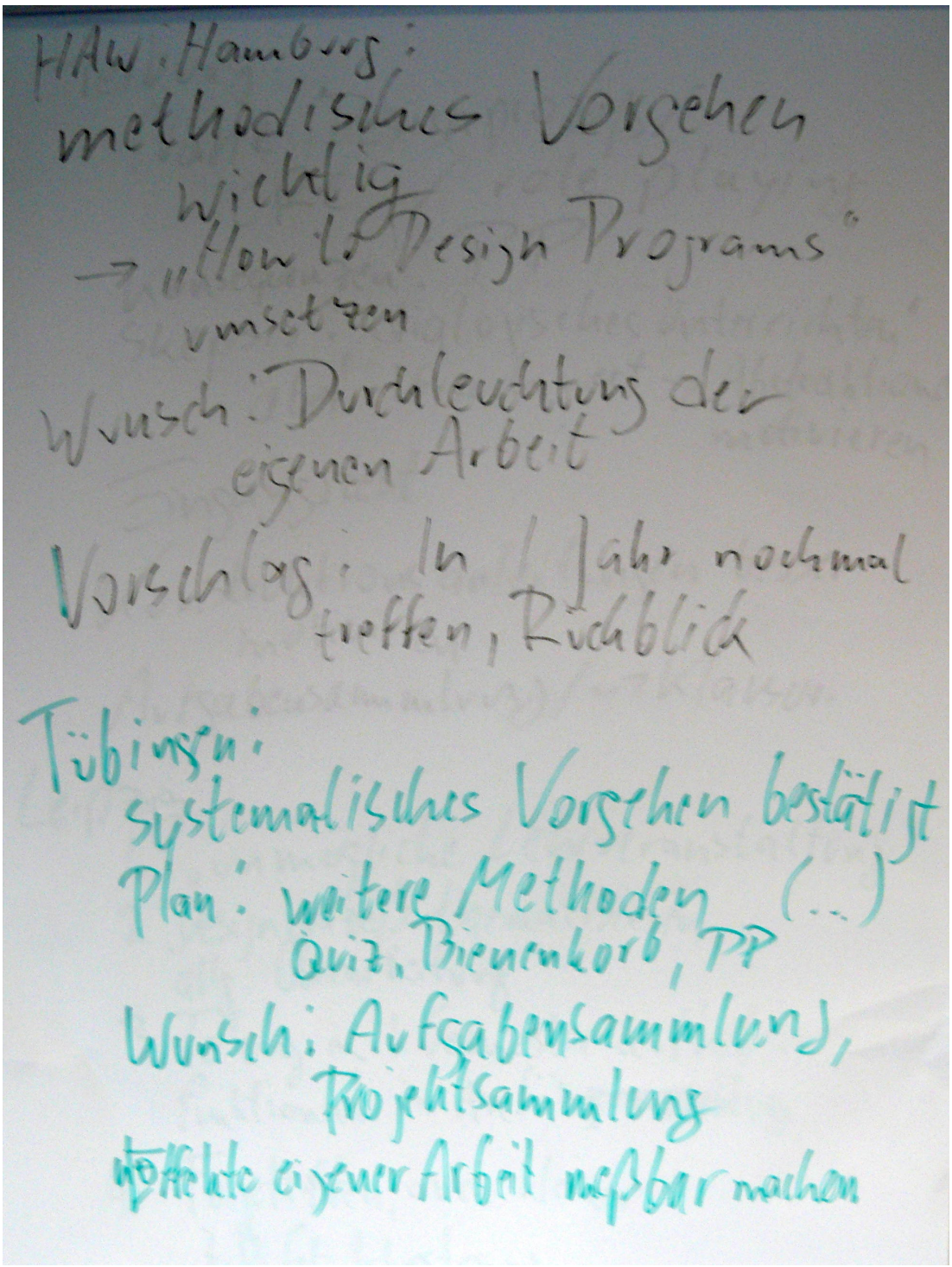
9 Erfolgsmessung

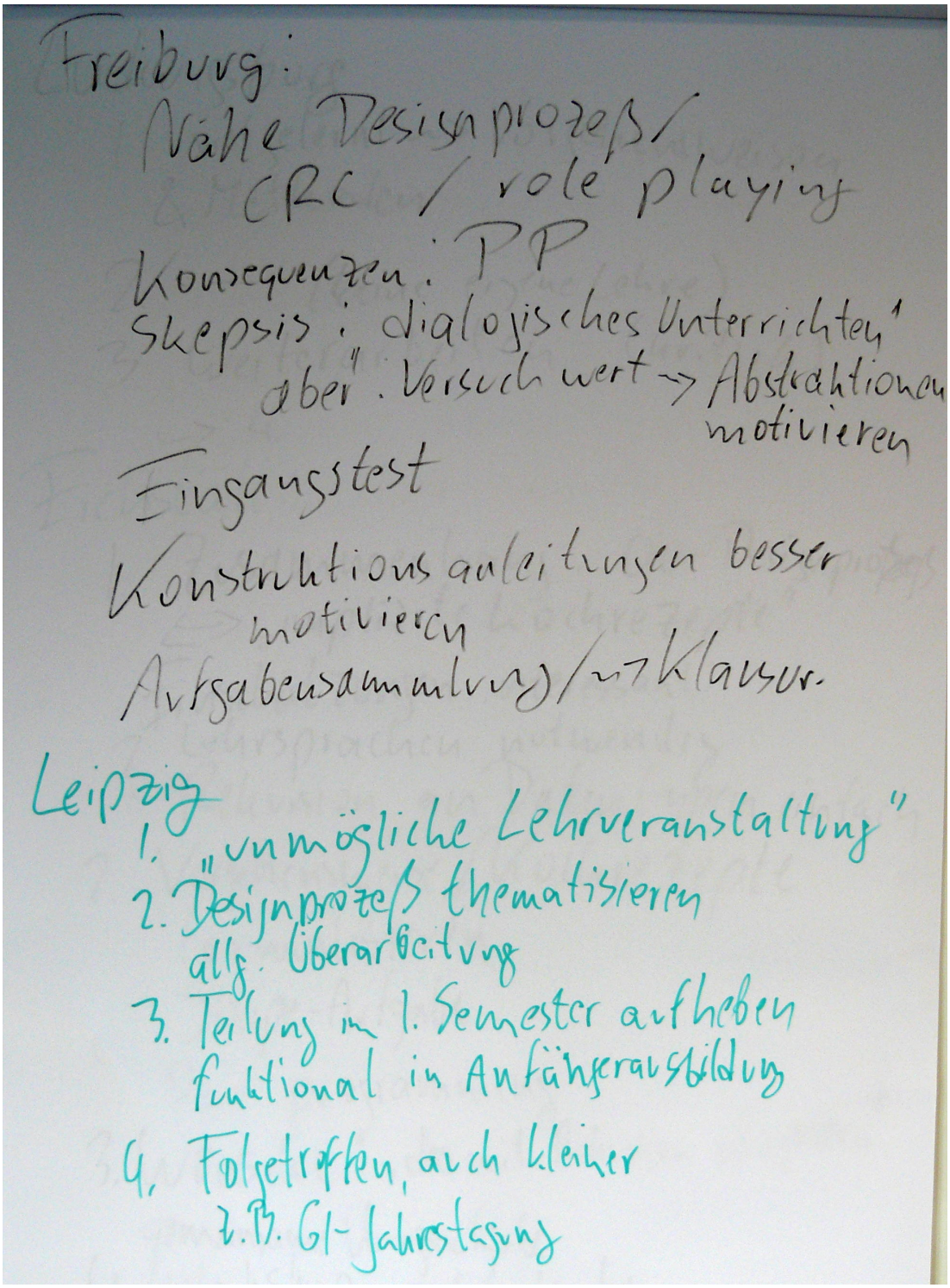


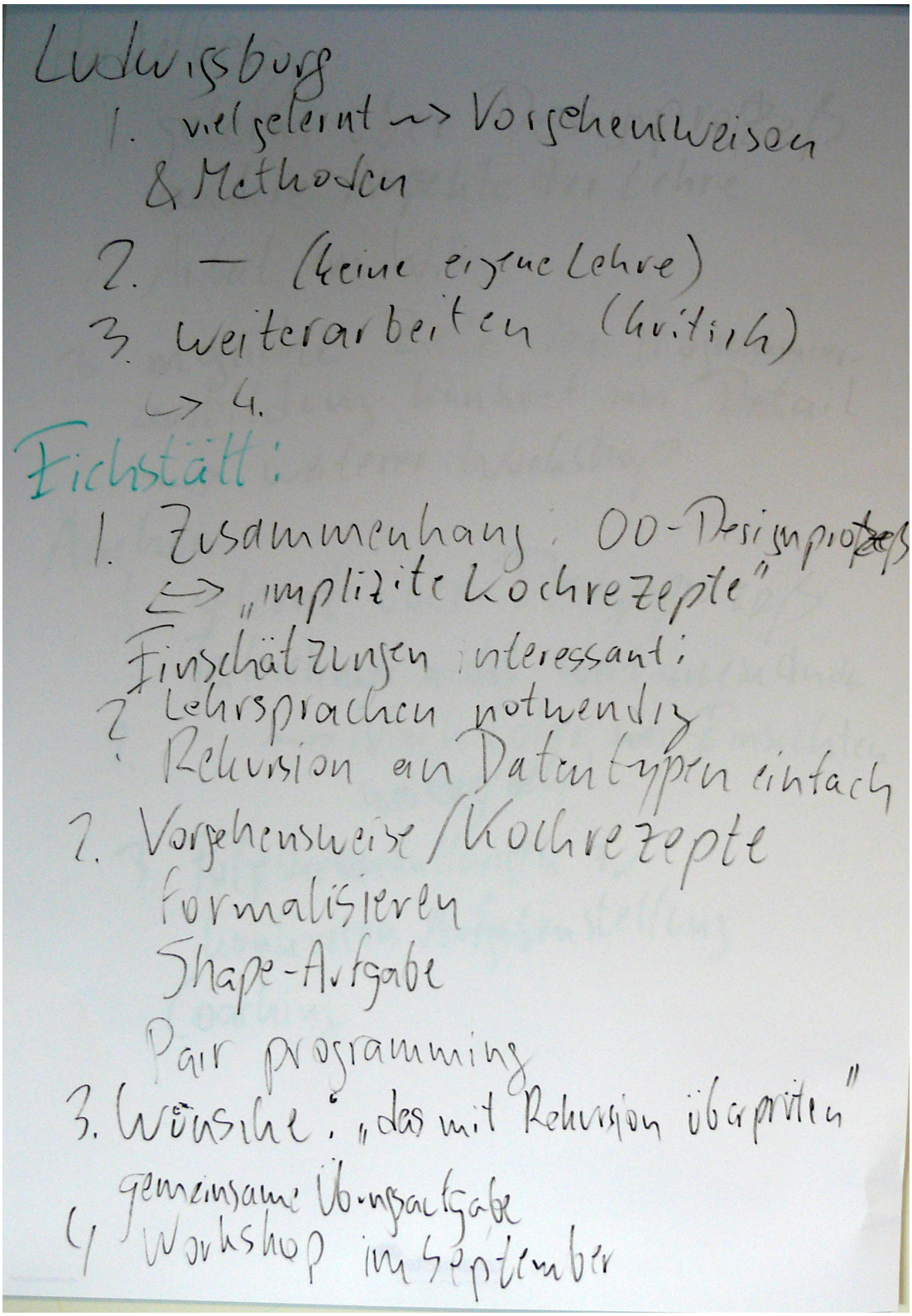
10 Zweigipflige Verteilung des Leisungsniveaus

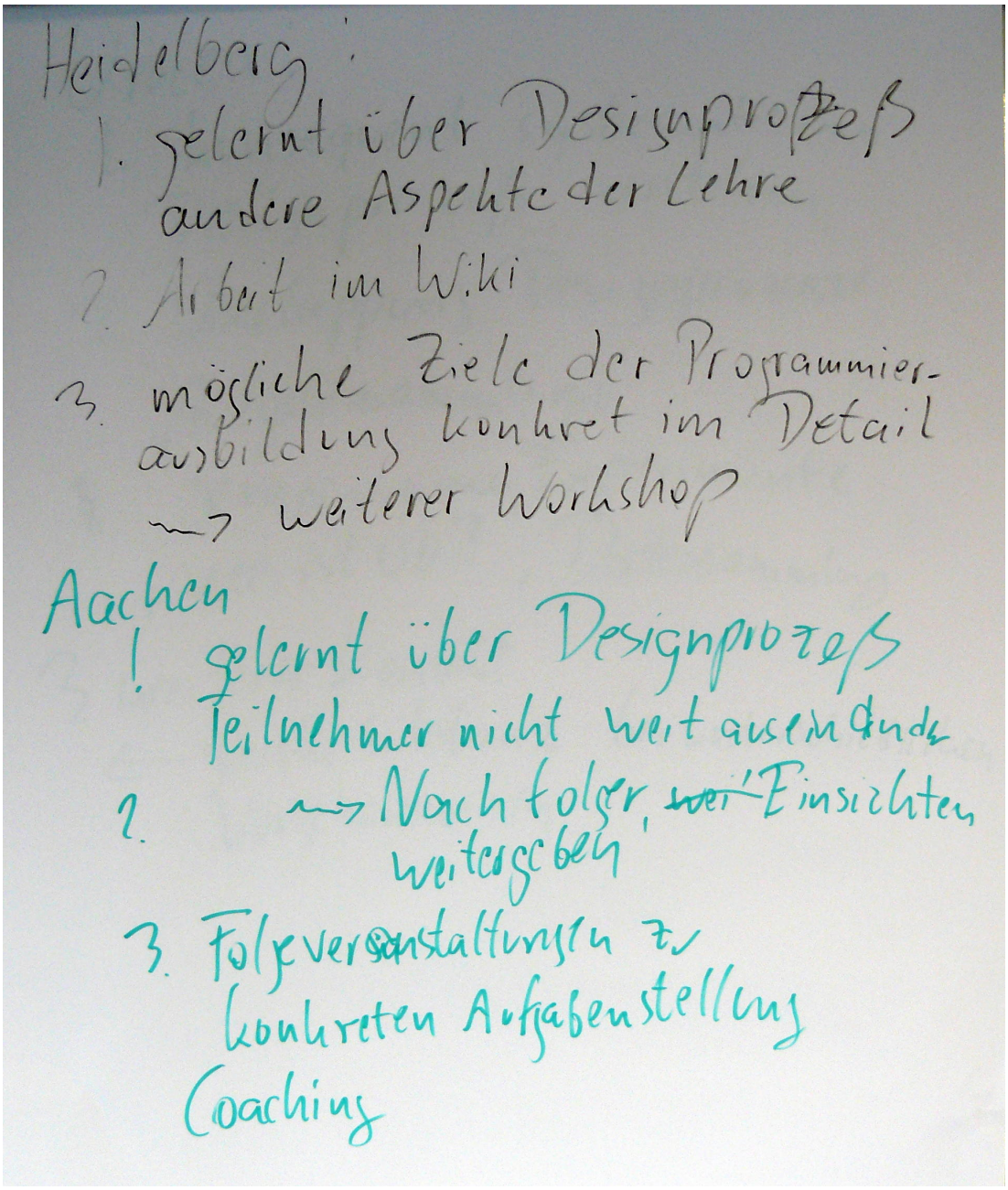


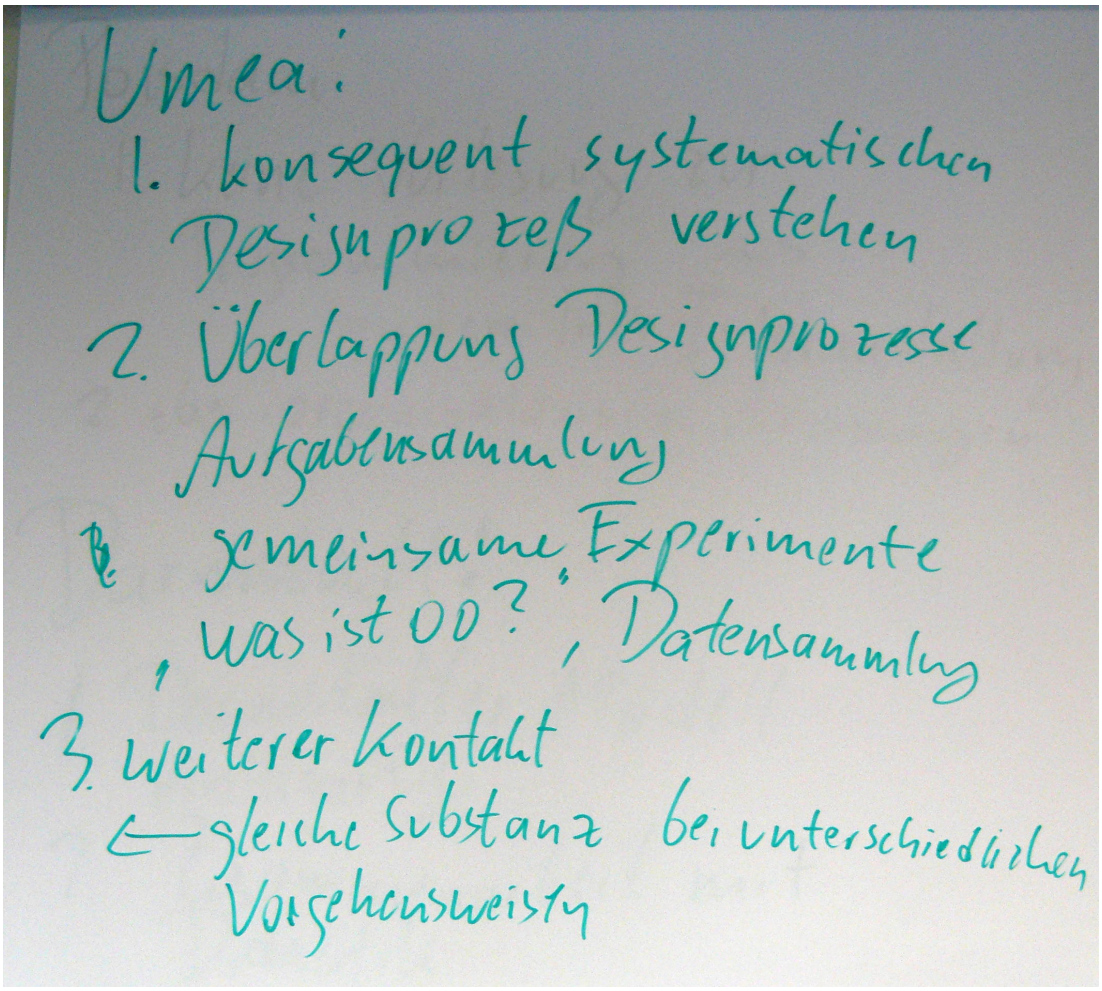
11 Einsichten und Konsequenzen aus dem Workshop

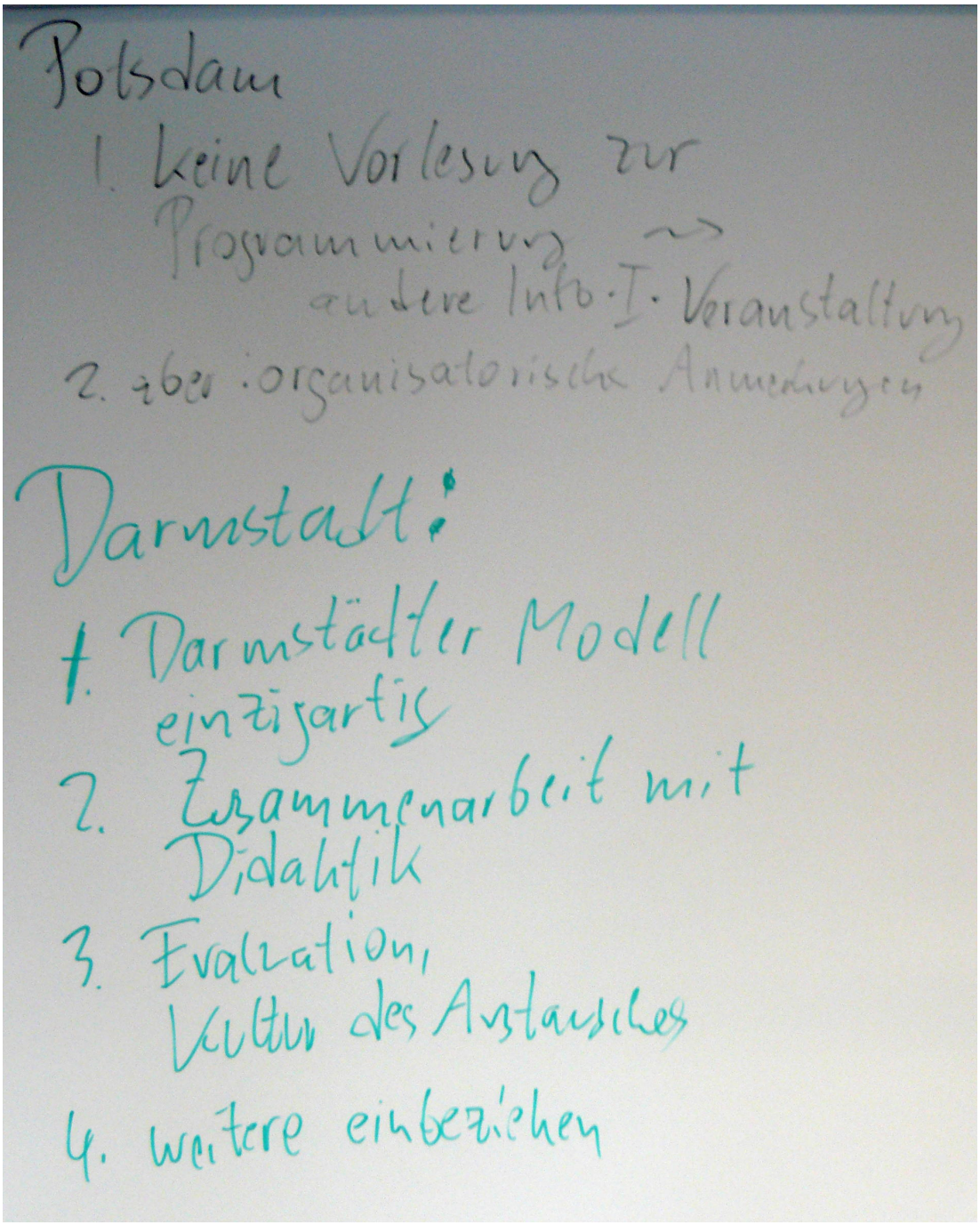



Freiburg:
Nähe Designprozess/
CRC / role playing
Konsequenzen: PP
Skepsis: dialogisches Unterrichten"
aber: Versuch wert -> Abstraktionen
motivieren
Eingangstest
Konstruktionsanleitungen besser
motivieren
Aufgabensammlung/ -> Klausur.
Leipzig
1. "unmögliche Lehrveranstaltung"
2. Designprozess thematisieren
allg. Überarbeitung
3. Teilung im 1. Semester aufheben
funktional in Anfängerausbildung
4. Folgetreffen, auch kleiner
z.B. GI-Jahrestagung


Ludwigsburg
1. viel gelernt ~> Vorgehensweisen & Methoden
2. — (keine eigene Lehre)
3. weiterarbeiten (kritisch)
~> 4.
Fichstätt:
1. Zusammenhang: OO-Designpr
<=> „implizite Kochrezepte"
Einschätzungen interessant
2. Lehrsprachen notwendig
3. Reflexion an Datentypen einfach
2. Vorgehensweise / Kochrezepte
formalisieren
Shape-Aufgabe
Pair programming
3. Wünsche: „das mit Reflexion überprüfen"
4 gemeinsame Übungsaufgabe
Workshop im September


Heidelberg:
1. gelernt über Designprozess
andere Aspekte der Lehre
2. Arbeit im Wiki
3. mögliche Ziele der Programmier-
ausbildung konkret im Detail
→ weiterer Workshop
Aachen
1. gelernt über Designprozess
Teilnehmer nicht weit auseinander
2. → Nachfolger, ~~vor~~ Einsichten
weitergeben
3. Folgeveranstaltungen zu
konkreten Aufgabenstellung
Coaching


Umea:
1. konsequent systematischen Designprozess verstehen
2. Überlappung Designprozesse
Aufgabensammlung
gemeinsame Experimente
„Was ist OO?", Datensammlung
3. weiterer Kontakt
← gleiche Substanz bei unterschiedlichen Vorgehensweisen



12 Mögliche Zusammenarbeit

Die Vorlesungen in Tübingen und Freiburg werden gemeinsam betrieben und weiterentwickelt: Dort existieren Lehrbuch, Software, Aufgabensammlung, Lehrbeispiele und anderes Material.

Michael Sperber bietet an, Vorlesungen zur systematischen Programmierungen nach dem Ansatz von *Die Macht der Abstraktion / How to Design Programs* zu begleiten und zu unterstützen.

13 White Paper

Ein „White Paper“ soll die im Workshop gewonnenen Einsichten dokumentieren. Folgende Teilnehmer haben sich zur aktiven Mitarbeit daran bereiterklärt:

- Jürgen Börstler
- Marcus Crestani
- Andreas Harrer
- Christian Spannagel
- Nicole Weicker
- Karsten Weicker