Besprechung Übung 2 & Vorstellung Übung 3

Software Qualität, FS09

04.05.2009

Reinhard Stoiber

Übung 2

- Ergebnisse: grossteils ausreichend, teilweise recht gut

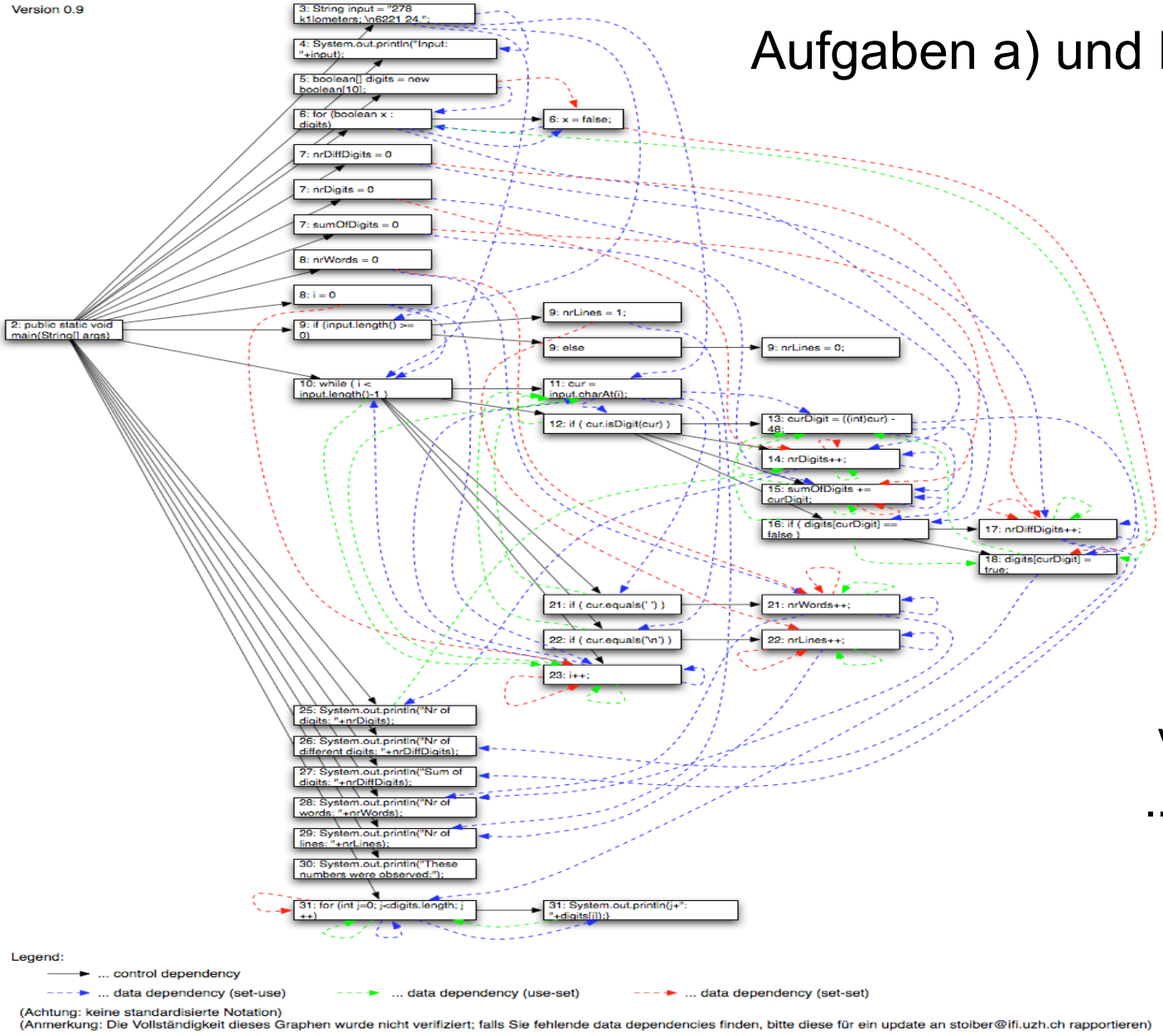
Teil A: Statische Analyse

- Ziel
  - Analyse der Daten- und Kontrollabhängigkeiten im Programmcode, ohne das Programm auszuführen
- Kriterien
  - Vollständige Beziehungen, korrekter Program Dependency Graph, Backward Slice abgeleitet, Probleme erkannt
- Lösungen
  - Oft wurden Beziehungen nur zwischen Zeilen angegeben, nicht aber zwischen den tatsächlichen Statements

Achtung: im gegenen Code gibt es einen konkreten Input;

Beziehungen zw Statements (PDG)

Aufgaben a) und b)



Dieser PDG zeigt die Daten- und Kontroll-abhängigkeiten

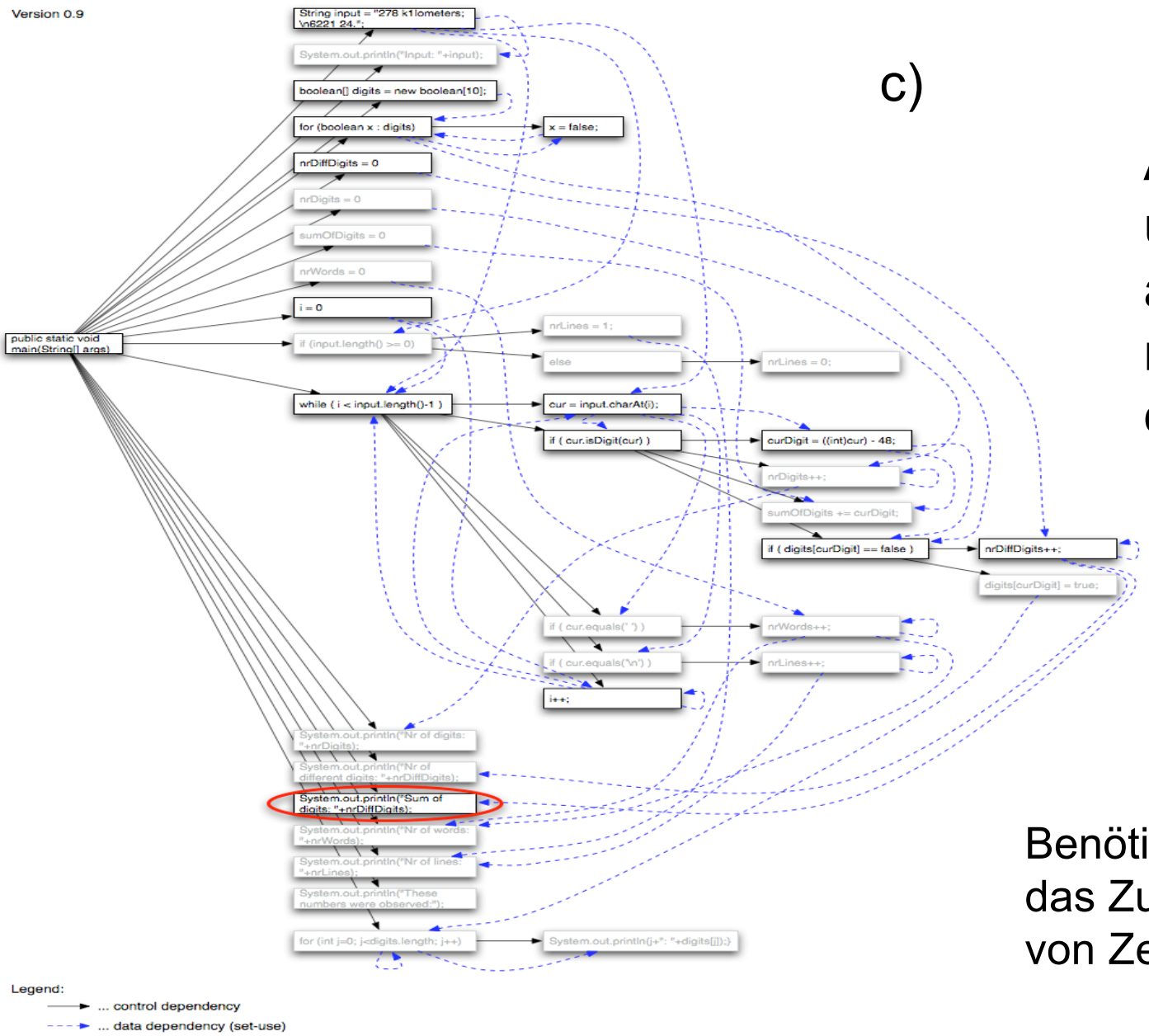
set-use

vorher nachher

...im Programmablauf

Anm.: **konkreter Input**

PDG: Backward Slice



c)

Alle Kontroll- und Daten-abhängigkeiten rückwärts durchlaufen

Benötigte Statements für das Zustandekommen von Zeile 27.

Problemerkennung mittels statischer Analyse

d) Probleme die erkannt werden können

- Ein else-Zweig (Zeile 9) wird nie erreicht
  - `if(input.length() >= 0)` kann nie ‘false’ sein
- sumOfDigits wird nach dem letzten ‘set’ nicht mehr verwendet (Zeile 24: kein set-use mehr)

- Generell: Code der nie aufgerufen wird, bzw. das restliche Programm nicht beeinflusst ...

Teil B: Rekonstruktion von Fehlern

- Ziel
  - Durch welche äusseren Einflüsse kann es schwierig werden einen einmal entdeckten Fehler zu reproduzieren?
- Kriterien
  - Einflussfaktoren und konkrete Beispiele genannt; Möglichkeit zur Reproduktion genannt
- Lösungen
  - Meist gut; teilweise fehlten konkrete Beispiele

Beispiele: Einflussfaktoren

- Verwendung von Zufallszahlen
  - Nur bestimmte Zufallswerte verursachen Fehler
  - Abhilfe: Aufzeichnung der Zufallswerte in einem Log und Verwendung dieser zur Rekonstruktion
- Persistente Daten
  - Falsche Datenbankeinträge verursachen Fehler
  - Nur wenn solche Einträge verwendet werden kommt es zum Fehler
  - Abhilfe: durch Logging die Anwendung die falsche Einträge schrieb identifizieren und korrigieren

Teil C: Debuggen

- Ziel
  - Systematische Fehlereingrenzung und Korrektur
- Kriterien
  - Systematisches Finden des einfachsten fehlerhaften Input; Variablenbelegung Soll; Hypothesen geprüft; Var.belegung Ist; Hypothesen verfeinert; Defekt korrigiert;
- Lösungen
  - Meist gut gelöst; einfachster Input nicht immer gefunden;

Finden des einfachsten Inputs

a)

Input nach erstem Schritt:

"Quicksort (von engl. quick - schnell, to sort - sortieren) ist ein schneller, rekursiver, nicht-stabiler Sortieralgorithmus, der nach dem Prinzip Teile und herrsche (lat. Divide et impera!, engl. Divide and conquer) arbeitet. Er wurde ca. 1960 von C. Antony R. Hoare in seiner Grundform entwickelt und seitdem von vielen Forschern verbessert."	Fehler

Input nach dem 2. Schritt:

"Quicksort (von engl. quick - schnell, to sort - sortieren) ist ein schneller rekursiver, nicht-stabiler Sortieralgorithmus, der nach dem Prinzip Teile und herrsche"	Fehler
"lat. Divide et impera!, engl. Divide and conquer) arbeitet. Er wurde ca. 1960 von C. Antony R. Hoare in seiner Grundform entwickelt und seitdem von vielen Forschern verbessert."	Fehler

Input nach dem 3. Schritt:

"Quicksort (von engl. quick - schnell, to sort - sortieren) ist ein schneller"	Fehler
"rekursiver, nicht-stabiler Sortieralgorithmus, der nach dem Prinzip Teile und herrsche "	OK
"(lat. Divide et impera!, engl. Divide and conquer) arbeitet. Er wurde ca. 1960 von C. "	Fehler
"Antony R. Hoare in seiner Grundform entwickelt und seitdem von vielen Forschern verbessert."	OK

Input nach dem 4. Schritt:

"Quicksort (von engl. quick - schnell,"	OK
" to sort - sortieren) ist ein schneller "	OK

geviertelt

Input nach dem 5. Schritt:

„Quicksort (von engl. quick"	OK
„ - schnell, to sort -„	Fehler
"sortieren) ist"	OK
„ein schneller"	OK

...

Input nach Schritt x:

„- -„	Fehler

einfachster fehlerhafter Input

(aus einer Abgabe)

Korrekte Variablenbelegung (Erwartete Werte)

- Hypothese 1

b)

Variable	Tatsächlicher Wert	Erwarteter Wert
start		0
end		1
untergrenze		1
obergrenze		1
vergleichemit		0
elemente		a, a

(aus einer Abgabe)

Erwartete Werte für ein korrektes Programm ...

Korrekte Variablenbelegung (Tatsächliche Werte)

- Hypothese 1

c)

Variable	Tatsächlicher Wert	Erwarteter Wert
start	0	0
end	1	1
untergrenze	1	1
obergrenze	1	1
vergleichemit	0	0
elemente	a, a	a, a

Die tatsächlichen Variablenwerte können mit dem Eclipse Debugger einfach festgestellt werden.

(aus einer Abgabe)

Konkrete und Erwartete Werte stimmen überein

→ Fehler trifft nicht zu; Hypothese wird verworfen

Verfeinern der zutreffenden Hypothese

d)

- Hypothese 5 trifft zu
  (je nach Betrachtungsweise auch Hyp. 2 möglich)
- Verfeinern von Hypothese 5
  - z.B. Hypothese 5.1: “der erste Rekursionsaufruf wird unendlich oft aufgerufen, wenn ein Wort zwei mal vorkommt”
    (aus einer Abgabe)
    - trifft zu
  - weiter, Hypothese 5.1.1: “der erste Rekursions-aufruf wird unendlich oft aufgerufen, weil bei Wortgleichheit die Obergrenze nicht dekrementiert wird”
    - trifft zu (mit erwarteten und tatsächlichen Werten zeigen)
    - Der Defekt ist bereits recht weit eingegrenzt

Korrektur des Defekts

- entweder hier weiter dekrementieren

d)

```
//zaehle obergrenze herunter solange die elemente groesser sind
for (; (obergrenze >= untergrenze) && // Debug-Punkt 3
(qc.kleinerAls(elemente[obergrenze],
        elemente[vergleichemit]) > 0); --obergrenze );
```

>=

- oder beim Rekursionsaufruf

-1

```
quicksort(elemente, start, obergrenze, qc);
```

Analyse des Programmierstils

e) In diesem Beispiel

- Die for-Schleifen sind nicht ideal, da nichts deklariert wird; besser: while
- Eine explizite Abbruchbedingung macht die innere while-Schleife (`while(true) {...}`) lesbarer
- ...

Übung 3

- Vorstellung der Übung 3
- Produktqualität definieren, bewerten und erreichen

Übung 3 - Allgemeines

- Daten
  - Ausgabe: 04.05.2009
  - Abgabe: 18.05.2009, 24h
  - Besprechung: 25.05.2009
- Abgabe, Formales
  - Abgabe an stoiber@ifi.uzh.ch
  - Es kann in den Computerräumen des IFI gearbeitet werden
- Gruppen
  - Die Übung wird in 2er Gruppen ausgearbeitet

Teil A: Produktqualität definieren

- Gegebenes Szenario
  - Weiterentwicklung von Bugtracking Software
  - Sie wollen die Qualität eines Produktes systematisch verbessern und neue Benützer und Marktanteile gewinnen
- Überblick zu Bugtracking Siehe auf Wikipedia: http://de.wikipedia.org/wiki/Bug-Tracker

ISO/IEC 9126-1 Qualitätsmodell

- Welche typischen Bedürfnisse haben Benützer einer Bugtracking Software?

- Verwenden Sie dazu das Modell ISO/IEC 9126-1 “quality model for quality in use”
  - Charakteristiken: effectiveness, productivity, safety, satisfaction
- Finden Sie Qualitätsziele und -anforderungen um diese Charakteristiken konkret zu erfüllen

Qualitätsanforderungen: Risiken und Werte

- Quantifizierung
  - genauere Qualitätsanforderungen
- Risiko
  - hohes Risiko erfordert präzise Anforderungen
- Wert
  - hohe mögliche Wertschöpfung kann durch präzise Anforderungen besser ausgeschöpft werden
- Nicht alles ist quantifizierbar; im Fall andere Methoden wählen ... (z.B. prototyping, ...)

Wichtigkeit / Gewicht von Qualitätsanforderungen

- Bewerten Sie für jede Qualitäts-anforderung die Wichtigkeit für die Produktqualität
- Werte- und Risikoabschätzung helfen hier

Konkrete Produktmerkmale definieren

- Die Produkte: http://de.wikipedia.org/wiki/Bug-Tracker
- Norm: ISO/IEC 9126-1 “quality model for external and internal quality”
- Benutzerbedürfnisse
  - functionality, usability, efficiency, maintainability, portability
- Für jedes dieser Bedürfnisse mindestens drei konkrete Produktmerkmale finden/definieren

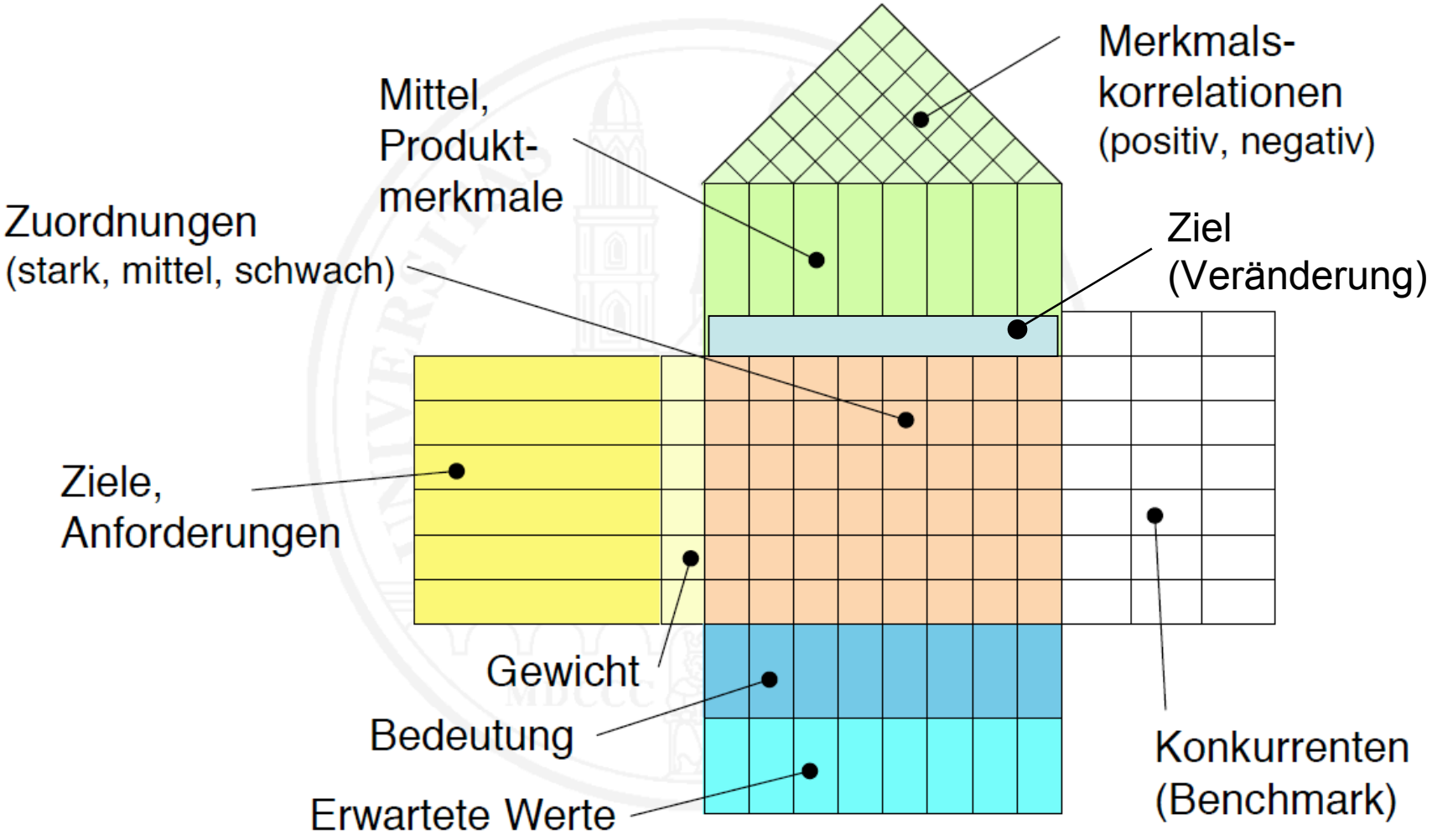
Teil B: Produktqualität bewerten und erreichen

- In Teil A wurden die Kriterien für Produkt-qualität bereits definiert
- Mit der Methode Quality Function Deployment (QFD) soll nun ein Qualitätshaus erstellt werden
- Ziele
  - Bewertung der Produktqualität
  - Systematische Verbesserung der Produktqualität

Qualitätshaus (1)

- Verwenden Sie das gegebene Excel Template
- Übertragen Sie alle Ihre Qualitätskriterien aus Teil A in das Qualitätshaus
- Evaluieren Sie damit zwei konkrete existierende Bugtracking Lösungen

Qualitätshaus (2)



Evaluieren der Produktqualität

- Wie stark helfen die Produktmerkmale das Benutzerbedürfnis zufriedenzustellen?
  - Skala: 1 - schwach, 3 - mittel, 5 - sehr stark
- Beeinflussen sich die Produktmerkmale gegenseitig?
  - [leer] - kein Einfluss; (+), (++) - positive Korrelation; (-), (--) - negative Korrelation
- Wie gut erfüllen die Konkurrenten die Qualitätsmerkmale?
  - Skala: 1 - wenig, 3 - mittel, 5 - sehr gut
- Welcher der beiden Konkurrenten bietet die besser Lösung?

Produktqualität verbessern

- Durch Verbesserung welcher Produktmerkmale kann die Produktqualität von Bugtracking Lösungen (Quality in use) am effizientesten verbessert werden?
  - Achtung: Merkmalskorrelationen
- Zeigen und beschreiben Sie Ihre Strategie zur Qualitätsverbesserung ...

Reflektion: QFD, alternative Methoden

- Wie fanden Sie die Bewertung von Produktqualität mit dem Qualitätshaus?
  - Stärken, Schwächen, Risiken, Verbesserungsvorschläge
- Kennen Sie eine alternative Methode(n)?
  - Beschreiben Sie diese und skizzieren Sie die wesentlichen Unterschiede zum QFD ...

Abgaben

- Ihre Lösungen zu den Aufgaben A und B als PDF, und Ihr Qualitätshaus im Excel und PDF Format

- Danke für die Aufmerksamkeit.