

Dies ist keine „Musterlösung“, sondern eine gute von vielen möglichen Lösungen. Kommentare, die nicht Teil der Lösung sind, sind kursiv gesetzt.

**Aufgabe 11-1:**

1. Statische Modelle (z.B. in Form von Klassendiagrammen) ergeben nur wenige Möglichkeiten zum Testen. Man kann die dort formulierten Invarianten (z.B. Multiplizitäten oder OCL-Ausdrücke) bei den laufenden Tests stetig (automatisch) prüfen (z.B. mittels Assertions), aber Testfälle entstehen daraus nicht direkt. Die kann man aber aus allen *dynamischen* Modellen entwickeln, z.B. Szenarien anhand von Sequenzdiagrammen (ergeben meist Akzeptanztests), Systemreaktionen bei Zustandsübergängen anhand von Zustandsdiagrammen (state charts) oder Ereignisfluss von Anwendungsfällen anhand Use-Case-Diagrammen(+Doku).

2. Man nimmt an, dass die Voraussetzungen „Student angemeldet“ und „Veranstaltung ausgewählt“ erfüllt sind. Das kann man machen, weil deren Erfüllung andere Tests abdecken müssen. Dann geht man strategisch so vor, dass man das normale Szenario durchgeht und für jeden der Ausnahmefälle einen weiteren Fall einfügt. Also:

- \* Ein Fall, bei dem ein Termin frei ist.
- \* Ein Fall, bei dem kein Termin frei ist.
- \* Ein Fall, bei dem die Veranstaltung von diesem Student belegbar ist.
  - \* Einmal mit einem Studenten, der schon zu viele Leistungspunkte gebucht hat.
  - \* Einmal mit einem Studenten, der noch nicht zu viele Leistungspunkte gebucht hat.
- \* Ein Fall, bei dem die Veranstaltung von diesem Student nicht belegbar ist, weil:
  - \* Veranstaltung passt nicht zum Studienverlauf.

Jedes mal muss danach geprüft werden, ob die Buchung korrekt statt gefunden hat oder korrekt nicht.

3.a. Der Fachbereich hat 1700 eingeschriebene Studierende und 200 potenzielle Dozenten + Sekretärinnen/Verwaltungsangestellte, die mit der Software zu tun haben könnten (einigermaßen sichere Zahlen) und (schätzungsweise) 500 Nebenfachstudierende, die ja auch was mit dem Prüfungsverwaltungssystem zu schaffen hätten. Die kritischste Zeit ist vermutlich der Semesteranfang, wenn so ziemlich jeder das System benutzt. Mal angenommen, an einem Spitzentag greifen alle Leute zu (etwa 2000), jeder von diesen liest vielleicht 10 Seiten, was in einer Minute (ein Tag hat 8h \* 60min = 480 davon, also etwa 500) etwa  $(2000/500)*10 = 40$  parallele (alles innerhalb einer Minute wird hier als parallel gesehen!) Zugriffe bedeutet. Dabei sind schreibende (Neueingabe oder Anmelden) und lesende Zugriffe möglich. Schreibende Zugriffe sind aber deutlich seltener, so dass ihre erhöhte Komplexität kaum ins Gewicht fällt.

3.b. Auch bei 40 gleichzeitigen Zugriffen sollte die Antwortzeit des Systems bei einer üblichen, vorher aber nicht durchgeführten Leseanfrage (z.B. Anzeigen einer Veranstaltung, die nicht im Client- oder Server-Cache sein darf) 5 Sekunden und bei einer Schreibanfrage (z.B. Anmelden bei einer Veranstaltung) 10 Sekunden nicht überschreiten. *Das sind durchaus beachtliche Anforderungen.*

3.c. Man sollte 20 gleichzeitige Zugriffe simulieren, d.h. kleine Clients (Treiber) schreiben, die so tun, als würden wirklich mehrere Benutzer gleichzeitig zugreifen, vielleicht die Hälfte schreibend, die andere Hälfte lesend, bei derselben ausgewählten Veranstaltung. Was man dabei aber nicht im Griff hat ist (a) die sonstige Belastung des (Inter-)Netzes und evtl. (b) die sonstigen Anwendungen und deren Last auf dem Server, auf dem die Prüfungsverwaltung läuft und dem Client. Hier sollte also für eine realistische Umgebung gesorgt werden.

**Aufgabe 11-2:**

2. Fast alle sind nicht automatisch prüfbar, aber z.B. 8 doch.

3. Mindestens alle unter dem Abschnitt Multithreading

4. Tests gehen nur für ausführbare Artefakte; Defekte/Mängel werden bei Durchsichten sofort lokalisiert; Man kann zusätzliche Arten von Mängeln entdecken (z.B. Kodierrichtlinien); Man hinterfragt dabei auch das korrekte Verständnis der Anforderungen/Spezifikation und deren vollständige Umsetzung; Man kann dabei nebenbei neue Mitarbeiter in ein Gebiet einarbeiten.

**Aufgabe 11-3:**

Die Argumentation ist im Wesentlichen psychologisch: Ein Mensch kommt einfach durcheinander, da er bei GOTOs immer tiefere Aufrufstufen zu merken hat beim Verstehen eines Programms. Er spricht nicht vom Kodieren selbst.

*Er hat übrigens nur den psychologischen Grund beschrieben ("our powers to visualize process evolving in time are relatively poorly developed", "terrible hard", "extremely complicated"), aber wenig zur empirischen Haltbarkeit gesagt ("familiar with the observation", "I discovered").*