Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Dynamisches Testen

Phillip Raffeck, Simon Schuster, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

Wintersemester 2021

Spezifikation einer Warteschlange I

Die Warteschlange soll folgenden Anforderungen genügen: Es soll möglich sein eine beliebige Anzahl von Warteschlangen zu erstellen und die maximale Anzahl der Elemente einer solchen Warteschlange soll nur durch den verfügbaren freien Speicher und die Zahl der Prioritätsebenen begrenzt sein. Jede Prioritätsebene soll innerhalb einer Warteschlange maximal einem Element gleichzeitig zugewiesen sein. 0 entspricht hierbei der höchsten Priorität. Die Warteschlange soll nicht-vorzeichenbehaftete 64 Bit-Ganzzahlen verwalten.

Versucht der Benutzer ein Element mit einer Priorität einzufügen, die in der Warteschlange bereits vergeben ist, so ist dies ein Fehler. Dieser soll nicht durch eine Ausgabe auf dem Bildschirm, sondern durch einen Rückgabewert angezeigt werden. Geht während des Aufrufs einer Warteschlangenoperation der Hauptspeicher aus, so ist dies ein Fehler,

Spezifikation einer Warteschlange II

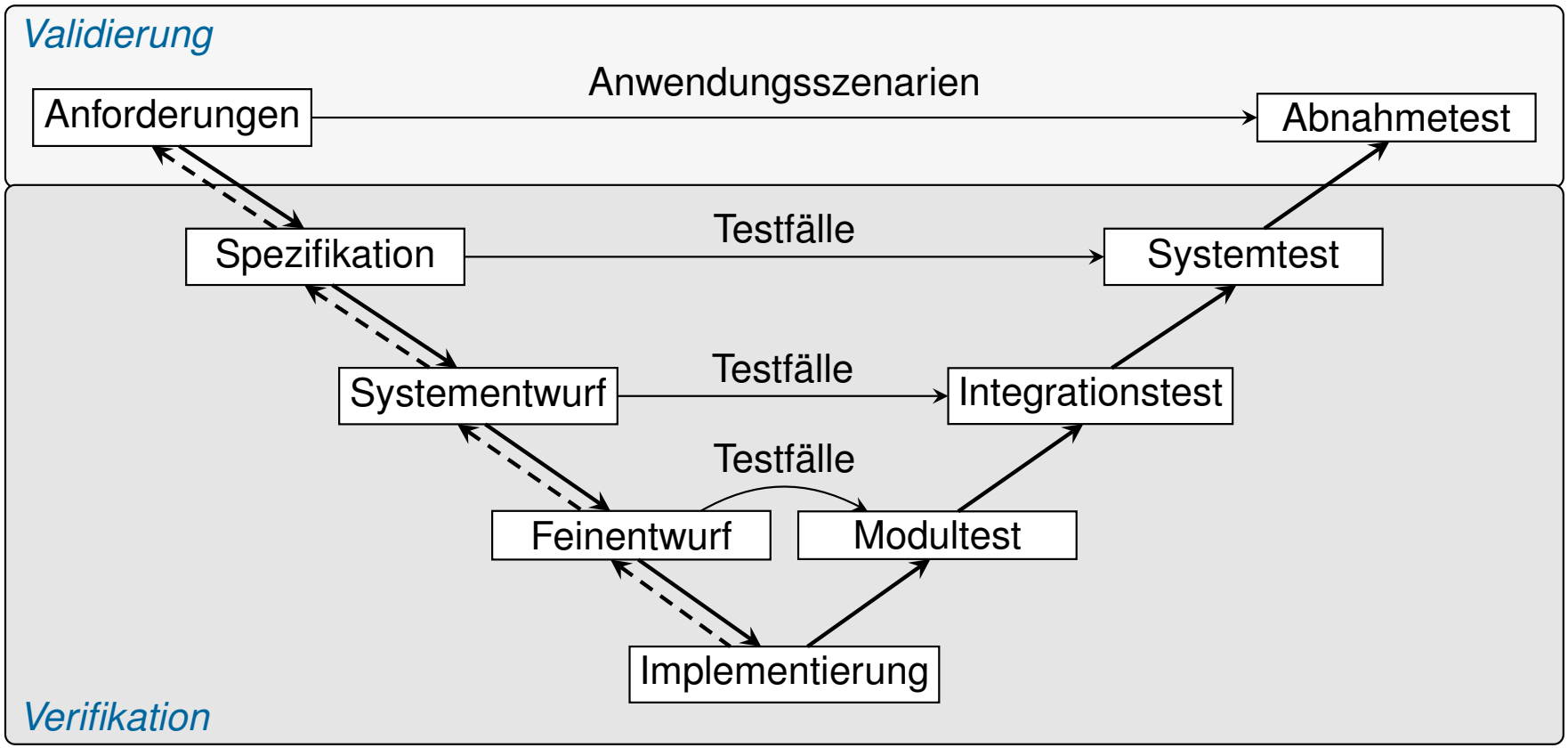
über den der Benutzer über die API benachrichtigt wird. Die Reaktion auf den Fehler ist dem Aufrufer überlassen.

Die Warteschlange soll die Möglichkeit bieten, das höchstpriore Element zu entfernen und hierbei den Wert dieses Elements dem Benutzer zugänglich machen. Ferner soll es möglich sein, die Anzahl der Elemente, die sich derzeit in der Warteschlange befinden zu erfragen.

Im Fehlerfall soll sich die Warteschlange allgemein gutmütig verhalten, d. h. ein Speicherzugriffsfehler oder nicht definiertes Verhalten sind beispielsweise nicht akzeptabel. Sofern das Verhalten der Warteschlange vom Laufzeitsystem abhängig ist, soll dies dokumentiert werden.

Nebenläufige Operationen auf der Warteschlange müssen **nicht** unterstützt werden. Es ist also keine Synchronisation erforderlich.

Einordnung in den Entwicklungsprozess



Testen

- Erste Grundregeln:
  - Testbarkeit von vornherein einplanen
    - ⇝ Feingranulare Testfälle
    - ⇝ *Separate Testfälle für jede einzelne Funktion!*
  - Teste Datentypen an ihren Wertebereichsgrenzen
    - → `INT16_MAX, INT16_MIN, ...`
  - Fehlerfälle explizit testen
  - *Minimale Testabdeckung*: erreichbarer Codeüberdeckung
- Hilfsmittel:
  - Überdeckungsmetriken
  - Automatisierte Testinfrastruktur

Vorsicht!

- Testfälle können nur die Anwesenheit von Fehlern zeigen
- Nicht deren Abwesenheit! (→ vgl. formale Verifikation)

⇝ Alle *Randfälle* erkennen und abdecken

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ `./build/review.tar.xz`: “Unsere Lösung” (Fester Termin!)
5. Entpacken nach `./review/libpriority_queue_alien.a`

To: i4ezsmux+projectX-dev@i4.cs.fau.de
Subject: Mögliche Fehler

Hallo liebes Dev-Team,

bei uns schlagen die folgenden Tests mit euerer Implementierung fehl:
- unordered_insert:
  Der Test fügt in inverser Reihenfolge ein und prüft den folgenden Satz der Spezifikation “...”
  Ausgabe: ...
- ... `make pack-review`, `ALIEN_TEST`, ... im CIP ausführen

Sag ⇝ Referenzumgebung für alle

Aufgabenstellung

- Aufgabenblatt ist dreigeteilt
- Nachbildung eines Entwicklungsprozesses
  - Entwicklungsteam
  - QA-Team
- **rechtzeitige** Abgabe der Einzelteile
- Teil 1
  - Eigene Implementierung
  - 28.05. – 12.06.
- Teil 2
  - Testen von **zwei** fremden Implementierungen
  - 04.06. – 19.06.
- Teil 3
  - Tiefergehende Analyse der eigenen Implementierung
  - 17.06. – 26.06.