Verteilte Systeme

Testtat

Bewertung und Hinweise

Stand 04.10.2019

Dozent: Patrick Jungk

Inhaltsverzeichnis

1 Allgemein

Die Aufgabenstellung richtet sich nach dem Inhalt der Vorlesung und ist in der jeweiligen Testtatsbeschreibung beschrieben.

Die Bewertung der einzelnen Lösungen richtet sich nach den folgenden Gesichtspunkten:

1. Analyse
2. Grobkonzept
3. Programmanteil
4. Testplan
5. Nachbetrachtung

Bei den einzelnen Gesichtspunkten sind unterschiedliche Kriterien ausschlaggebend (siehe Kapitel 2, Seite 2). Bei den Punkten 1 – 4 sind Gesichtspunkte aus der Vorlesung im Fach „Verteilte Systeme“ zu Berücksichtigen. Die Punkteverteilung richtet sich nach der Wichtigkeit im Bezug auf die Vorlesung. Dieses Dokument beinhaltet einige Tipps, die im Allgemeinen helfen sollen und im Einzelfall herangezogen werden können.

2 Punkteverteilung

2.1 Übersicht

	Analyse			
	Aufgabenteil	**Unterpunkt**	*Einzelpunkt*	**GESAMTPUNKTE** **95**
5	**Analyse**			
		3	**Aufgabenanalyse**	
			1	*Verständnis der Aufgaben*
			2	*Verteilung der Aufgaben*
		2	**Abgrenzung**	
			1	*Erkennen der Aufgabenteile*
			1	*Ausschließen von nicht zu behandelnden Faktoren*
35	**Grobkonzept**			
		10	**Architektur**	
			2	*Darstellung der Gesamtarchitektur*
			4	*Erläuterung der einzelnen Teilschichten*
			4	*Erläuterung der Zusammenhänge der einzelnen Teilschichten*
		10	**Verteilte Aspekte**	
			2	*Darstellung der unterstützten verteilten Aspekte*
			2	*Darstellung der verteilten Aspekte: Transparenz*
			2	*Darstellung der verteilten Aspekte: Synchronisation*
			2	*Darstellung der verteilten Aspekte: Nebenläufigkeit*
			2	*Darstellung der verteilten Aspekte: Kommunikation*
		5	**Konzeptioneller Ablauf**	
			3	*Darstellung des Ablaufs von einer Anfrage bis zur Antwort*
			2	*Darstellung des Anmeldens eines Zeitservers*
		5	**Schnittstellen**	
			1	*Nennung einzelnen Schnittstellen*
			2	*Beschreibung der Funktion der einzelnen Schnittstellen*
			1	*Beschreibung der Übergabe-/Rückgabeparameter der Schnittstellen*
			1	*Beschreibung der Fehlerbehandlung*
		5	**Beschreibung der Anpassung**	
			2	*Beschreibung der zusätzlichen Funktionalitäten*
			3	*Beschreibung der gelöschten/geänderten Funktionalitäten*
35	**Programm**			
		10	**Quellcode**	
			2	*Kommentierung der wichtigen Methoden*
			2	*Verständlichkeit der Kommentierung*
			2	*Sinnvolle Benennung der Methoden und Variablen*
			4	*Nutzung von Design-Pattern*
		10	**Umsetzung verteilter Aspekte**	
			2	*Umsetzung von Synchronisationsmechanismen*
			2	*Umsetzung von Nebenläufigkeit*
			2	*Transparenzumsetzung*
			4	*Fehlerbehandlung*
		5	**Lauffähigkeit**	
			1	*Programm startet*
			2	*Programm beendet bei Fehler definiert*
			2	*Programm läuft in einer der geforderten Laufzeitumgebungen/OS-Umgebungen*
		5	**Make-Anweisung/Generierungsanweisung**	
			1	*Anweisung komplett*
			1	*Voraussetzungen beschrieben*
			1	*Programm lässt sich kompilieren*
			1	*Programmabhängigkeiten beschrieben*
			1	*Programmabhängigkeiten lassen sich einbinden*
		5	**Startanweisung**	
			1	*Startanweisung ist komplett*
			1	*Systemvoraussetzung beschrieben*
			1	*Startanweisung ist verständlich*
			2	*Startanweisung führt zum fehlerfreien Start des Programms*
10	**Testplan**			
			2	*Testfälle sind definiert*
			2	*Testfälle sind sinnvoll*
			2	*Testfälle sind nachvollziehbar ausführbar*
			2	*Testfälle sind erfolgreich*
			2	*Ausreichend Testfälle definiert*
10	**Nachbetrachtung**			
			5	*Darstellung der Ergebnisse*
			5	*Einschränkungen/Systemgrenzen*

2.2 Analyse

Ausschlaggebend bei der Analyse ist die Aufgabenanalyse und die Abgrenzung der Aufgabe.

Bei der ***Aufgabenanalyse*** ist ausschlaggebend, dass das Verständnis für die Aufgabe dargestellt wird. Hierbei ist wichtig, dass das Ziel der Aufgabe und die Voraussetzungen dargestellt werden. Ebenso muss die Aufteilung der einzelnen Aufgabenanteile auf die teilnehmenden Studenten dargestellt werden. Gemeinsame Anteile müssen als solche dargestellt werden, wobei hierbei ebenso die Schwerpunkte der einzelnen Studenten dargestellt werden müssen.

Bei der ***Abgrenzung*** müssen die einzelnen Anforderungen herausgearbeitet werden, Haupt- und Nebenanforderungen zugewiesen und die Anforderungen den einzelnen Gruppenmitgliedern zugewiesen werden. Hierbei sind Faktoren auszuschließen, die zwar denkbar wären, aber nicht gefordert sind, insbesondere Faktoren aus dem Umfeld der verteilten Systeme. Die Anforderungen sind zu priorisieren und zu kategorisieren (sofern möglich).

TIPP: Eine Anforderungsmatrix hilft bei den Darstellung der Anforderungen, diese zu priorisieren, kategorisieren und die Anforderungen abzugrenzen. Eine solche Matrix kann dann ebenso für eine Abnahme herangezogen werden. Diese Matrix sollte mit dem Auftraggeber abgestimmt sein.

2.3 Grobkonzept

Das Grobkonzept sollte einen Überblick über die folgende Teilpunkte geben:

- die gewählte Architektur, Algorithmen und Pattern
- Unterstützung der geforderten Aspekte (hierbei verteilte Aspekte)
- konzeptionelle Kernabläufe
- Schnittstellen
- ggf. Anpassungen an einem bestehenden System (hierbei gefordert)

Bei der ***Architektur*** muss die Wahl begründet sein, und Alternativen ausreichend beleuchtet werden. Hierbei muss die Architektur als gesamtes dargestellt werden und die einzelnen Teilschichten beleuchtet werden. Wichtig hierbei ist der grobe Zusammenhang zwischen den einzelnen Schichten. Merke: die Komponenten der Architektur können auch disjunkt sein, je nach Aufgabenstellung, jedoch ist meist ein Zusammenhang dennoch nicht ganz auszuschließen. Ist eine Darstellung von Kundenseite als Vorlage vorhanden kann diese in die Gesamtarchitektur eingebettet werden, muss jedoch kritisch hinterfragt werden.

TIPP: Der Mensch denkt visuell, daher sind Diagramme und Ablaufpläne hilfreich. Den Auftraggeber bei der Erstellung zu konsultieren kann hilfreich sein und ein gemeinsames Verständnis für die Aufgabenstellung herstellen. Es können bei einem solchen Vorgehen frühzeitig Ausschlusskriterien erkannt werden (z.B. gegebene Sicherheitsaspekte/-voraussetzungen).

Sofern gewisse ***Aspekte*** gefordert sind, müssen diese bereits im Grobkonzept berücksichtigt und dargestellt bzw. analysiert werden werden gewisse Aspekte weggelassen, so muss dies begründet geschehen. Hierbei sind die Aspekte der verteilten Systeme zu berücksichtigen:

- Transparenz

- Synchronisierung
- Nebenläufigkeit
- Kommunikation

In wie weit ein Aspekt in der Aufgabenstellung eine Rolle spielt, hängt von den Anforderungen und den Rahmenbedingungen ab. Diese Bedingungen müssen für die Aspekte herausgearbeitet werden.

TIPP: Nicht alle geforderten Aspekte machen immer einen Sinn. Manche sind z.B. nur gefordert, weil diese Aspekt gerade „Mode“ ist. Wenn eine Aspekt keinen Sinn macht, z.B. weil die Kosten dadurch zu hoch werden, ein System zu langsam, bzw. zu Fehler trächtig ist, dann sollte der Auftraggeber mit einleuchtenden Argumenten auf die Sinnhaftigkeit hingewiesen werden. Die kann im Grobkonzept geschehen, sollte jedoch auf jeden Fall mit dem Auftraggeber vorab besprochen werden. Dabei sollte bedacht werden, dass auch die Argumente des Auftraggebers beachtet werden müssen. Oft muss ein Konsens gefunden werden (gemeinsam).

Kernabläufe müssen im Grobkonzept konzeptionell dargestellt werden. Hiermit wird ein Verständnis für die Prozesse, die ein System/eine Software unterstützen sollen, dargestellt. Teilnehmende Rollen sollten hierbei allgemein gehalten werden. Zu viele Details schränken in dieser Stufe zu sehr ein. Die Abhängigkeit der einzelnen Abläufe sollte ebenso dargestellt werden.

TIPP: Auch hier gilt: Der Mensch denk visuell. Die höchste Priorität bei den beschreibenden Diagramme sollten daher in der Verständlichkeit liegen. Zu Komplex verwirrt meistens nur.

Schnittstellen innerhalb des Systems müssen im Grobkonzept dargestellt werden. Dabei ist es wichtig folgende Punkte zu berücksichtigen:

- Funktion hinter der Schnittstelle: welche Funktionalität steckt hinter der Schnittstelle, und in wie weit beeinflusst diese die Gesamtarchitektur und die Prozesse.
- Übergabe-/Rückgabeparameter: welche Parameter werden erwartet, und zurückgegeben. Woher kommen diese Parameter und wohin müssen die Rückgabeparameter geliefert werden. Welche Programmanteile sind von den Parametern der Schnittstelle abhängig.
- Fehlerbehandlung: welche Fehler können im Umfeld der Schnittstelle geschehen und wie wird auf die Fehlerfälle reagiert.

Hierbei müssen die Schnittstellen spätestens im Feinkonzept detailliert beschrieben sein. Im Grobkonzept sollte jedoch auf die wichtigsten Schnittstellen eingegangen werden.

Sofern bei dem Programmanteil andere Systemanteile – Fremdanteile, sowie Anteile anderer Teammitglieder – zu berücksichtigen sind, müssen Schnittstellen zu diesen Anteilen beschrieben werden. Diese Schnittstellen müssen im Grobkonzept dargestellt werden.

TIPP: Eine Schnittstelle kann in jeglicher Form vorliegen, z.B. Schnittstelle zu dem Menschen, zu Peripheriegeräten etc..

TIPP: Innerhalb eines Teams sollten die Schnittstellen klar definiert werden. Während eines Projekts kann eine Schnittstelle sich ändern. Wichtig hierbei ist, dass bei jeder Änderung beide Seiten beteiligt sind.

Sollten sich während der Planung und Analyse der bestehenden Softwareumgebung,

die integriert werden soll, Änderungen ergeben, z.B. fehlerhafte Programmanteil in der bestehenden Umgebung oder Anpassungen aufgrund der gewählten Architektur, müssen diese beschrieben werden. Solche Beschreibungen sollten folgende Punkte beinhalten:

- Funktionalität, die geändert wurde (Beschreibender Name)
- Ist – Stand der Funktionalität
- Soll – Stand der Funktionalität
- Grund der Anpassungen
- Durchgeführte Anpassungen

TIPP: Anpassungen an bestehenden Funktionalitäten erfordern ein tiefes Verständnis derselben. Hierbei ist es oft erforderlich die Dokumentation der Funktionalitäten zu konsultieren und ggf. den Entwickler der ursprünglichen Funktionen einzubeziehen. Manchmal ist es auch effizienter die bestehenden Funktionalitäten zu verwerfen. Nichts unbegründet durchführen.

2.4 Programmanteil

Beim Programmanteil spielen folgende Punkte eine wichtige Rolle:

- Umsetzung der Forderungen
- Umsetzung der berücksichtigten bzw. zu berücksichtigenden Aspekte/Architektur
- Quellcode
- Lauffähigkeit
- Make-/Generierungsanweisung
- Startanweisung

Die Umsetzung der ***Forderungen*** steht primär im Vordergrund. Hierbei spielt ebenso die Integration der Forderungen in die Architektur ein gewisse Rolle, eine Besondere, sollte der Kunde auf eine gewisse ***Architektur*** in den Forderungen bestehen, bzw. diese erwarten. Hierbei wären dies die Forderungen aus dem Bereich der verteilten Systeme.

TIPP: Forderungen können sich mit der Zeit ändern, besonders wenn neue Erkenntnisse vorliegen. Hierbei können alte Forderungen auch verworfen werden, sofern der Kunde dem zustimmt. Vorsicht ist hierbei geboten, dass der Kunde nicht „zu viel“ bekommt.

TIPP: Sollten Forderungen nicht erfüllt werden können, sollte dies frühzeitig kommuniziert und begründet werden. Alternativen lassen sich unter Umständen gemeinsam finden.

Der ***Quellcode*** sollte gewissen Kriterien genügen, um spätere Erweiterungen/Anpassungen zu ermöglichen. Diese sind:

- Kommentierung wichtiger Methoden
  - Funktionsname
  - Funktionsbeschreibung
  - Eingabeparameter: Typ, Beschreibung, wichtige Eigenschaften (z.B. NotNull)
  - Rückgabeparameter: Typ, Beschreibung, wichtige Eigenschaften (z.B. NotNull), Rückgabeparameter im Fehlerfall (auch mehrere)
  - Autor (auch mehere)

    - Version (letzte Änderung)
    - Datum der Erstellung
    - Datum der letzten Änderung
- Verständlichkeit
    - Nachvollziehbarkeit (z.B. mittels Inline-Kommentierung
    - Überschaubarer Komplexitätsgrad
    - Sinnvolle Benennung von Methoden und Variablen (u.U. Camel-Case)
- Nutzung von „Design Pattern“ und gängigen Programmiermethoden

TIPP: Macht man sich unersetzlich dadurch, dass man schlecht kommentiert, so erntet man unter seinen Kollegen keinen Dank und bindet sich stark an ein Thema.

TIPP: Wählt man die Art der Kommentierung geschickt, kann man einen großen Teil der Entwicklerdokumentation durch Erstellungstools (z.B. doxygen) erzeugen lassen und spart sich dadurch Arbeit, was jedoch die Planung am Anfang nicht überflüssig machen sollte.

Nach Generierung des Programms sollte dieses dann ***lauffähig*** sein. Hierbei sollte da Programm starten, bzw. bei fehlenden Parametern darauf hinweisen, welche Parameter fehlen, und wie diese einzugeben sind. Nicht nur das Starten auch das Beenden sollte definiert geschehen. Besonders im Fehlerfall sollte sich das Programm möglichst kontrolliert beenden. Eine Exception in JAVA z.B. sollte abgefangen werden und behandelt sein. Ein System sollte nie in einen undefinierten Zustand wechseln, da hierbei weitere Systemkomponenten in Mitleidenschaft gezogen werden können (z.B. blockierte Ports).

Ist eine gewisse Umgebung gefordert, so muss die Software in dieser Umgebung laufen. Schon bei der Entwicklung sollte darauf geachtet werden.

TIPP: Ist keine Umgebung bzw. diese nur ungenau gefordert sollte eine Umgebung definiert und festgehalten werden. Die Umgebung sollte innerhalb der Entwicklung nicht geändert werden.

Der Kunde sollte in der Lage sein, dass Programm zu starten, hierzu sollte eine Startanweisung vorliegen, die folgende Kriterien erfüllt:

- Startanweisung ist komplett
- Systemvoraussetzungen sind beschrieben
- Startanweisung ist verständlich
- Startanweisung führt zum Start (bei richtiger Bedienung)

Hierbei kommt es dem Kunden entgegen, wenn er möglichst wenig zu tun hat. Alles, was nicht zumutbar ist, sollte bei der Entwicklung berücksichtigt werden. Es sollte auf keinen Fall den Benutzer sich selbst überlegen müssen, wie das Programm nun zu starten ist. Dies kann sogar zu ungewollten Effekten kommen, z.B. ungewolltes Eröffnen von Sicherheitslücken.

TIPP: Der Computer ist ein Hilfswerkzeug, wenn die Hilfe nicht gegeben ist, so kann der Benutzer theoretisch dagegen anklagen (Basis: BildscharbV, DIN9241, BITV), was im schlimmsten Fall auch den Entwickler treffen kann.

2.5 Testplan

Der Testplan stellt einen Bestandteil der finalen Abnahme dar. Dieser sollte in der Regel sehr umfangreich sein. Es sollten möglichst aussagekräftige Tests beschrieben sein, die sinnvoll sind. Die Tests sollten auch nachvollziehbar und reproduzierbar sein. In der Regel werden Test unter Berücksichtigung folgender Gesichtspunkte durchgeführt:

- Extremwert-Tests (min., max.)
- Durchschnittswert-Tests
- Erzeugen von HW-Fehlers (Kabel trennen, etc.)
- Lasttests
- Überlasttest
- Langzeit-Tests
- Tests der geforderten Aspekte
- Usability Tests
- etc.

Bei jedem Test sollten folgende Punkte erfasst werden:

- Nummer des Tests
- Bezeichnung des Tests
- ggf. Referenz auf Forderung
- Vorbedingungen
- Beschreibung der Testschritte
- erwartetes Ergebnis
- tatsächliches Ergebnis
- Nachbedingung
- Testergebnis (OK, NOK, Akzeptiert, Akzeptiert mit Nacharbeit bis)

Wichtig ist es, dass bei einem Test mit nicht erwartetem Ergebnis dieses genau notiert wird, sodass der Entwickler dies zu Fehlerbehebung nutzen kann.

TIPP: Tests dienen zu Aufdeckung und Behebung von Fehlern und nicht dazu, einem Kollegen schlechtes anzutun. Es ist hilfreich zusammen an einer Lösung zu suchen, anstatt nur zu kritisieren.

2.6 Nachbetrachtung

Die Nachbetrachtung dient hierbei zur Reflexion über die Arbeit. Die beinhaltet:

- Betrachtung der Durchführung
- „Lessons learned“
- Kritische Betrachtung der Ergebnisse
- Teamwork/Teamerfahrungen
- Nicht behandelte Aspekte (die sinnvoll gewesen wären)

Ebenso sollte hierbei das System beleuchtet werden, dies beinhaltet:

- Systemgrenzen

- Systemperformance
- Systemerweiterungen
- Kritische Nachbetrachtung der gewählten Architektur
- Systemprobleme

Ziel ist es das gelernte zu Verinnerlichen und Verbesserungen zu erkennen.