Learning Outcome der Veranstaltung Softwaretechnik 1

Stefan Bente, Stand 22.03.2017

Inhalt

1 Kompetenzprofil der Veranstaltung ST1

	ST1_Ziel			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	eine Softwarearchitektur für eine gegebene Aufgabenstellung gemäß erprobter Patterns und Architekturstile iterativ	erarbeiten	3	
und	in jeweils passender UML-Notation	dokumentieren	3	
vorausgesetzt, ich kann	Sinn und Gegenstand von Softwaretechnik allgemein, und Architekturarbeit im Besonderen,	erklären	2	ST1_Ziel.SWT_allg
	Softwaretechnik in den SW-Lebenszyklus	einordnen	1	
	die Grundbegriffe von IT-Architektur	erklären	2	
	die wesentlichen Komplexitätstreiber von Software, für die Architektur ein Management-Mittel darstellt,	benennen	2	ST1_Ziel.KompTr
	die generellen Vor- und Nachteile von UML als Modellierungssprache	benennen	2	ST1_Ziel.UML_VN
	die sechs für die Praxis wichtigsten UML-Diagramme	aufzählen	1	ST1_Ziel.6UML
	und diese in das Raster aus Verhaltens-, Struktur-, Nutzer-, Entwicklungsperspektive	einordnen	2	ST1_Ziel.Persp
indem ich	durch Use Cases, fachliches Datenmodell und Clustering eine erste Komponentenstruktur	definiere	4	KompStr
	die Geschäftsprozesse meines Systems (samt der Schlussfolgerungen für mein Komponenten-Design)	modelliere	4	GPKomp
	das Verhalten der Komponenten durch Modellierung der Geschäftsobjekt-Zustände näher	spezifiziere	4	GOZust
	Datenhaltung und Datenzugriff durch die Weiterentwicklung des fachlichen zum logischen Datenmodell	definiere	4	LogDM
	die Schnittstellen der Komponenten durch Interfaces	spezifiziere	4	Schnitt
	den Datenaustausch der Komponenten durch Sequenzdiagramme	definiere	4	KompSeq
und abschließend				
so dass ich	das entsprechende IT-System im Team in nachhaltiger technischer & fachlicher Qualität teil- / gesamtverantwortlich umsetzen kann.			

1.1 Definition einer (initialen) Komponentenstruktur

	KompStr			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	durch Use Cases, fachliches Datenmodell und Clustering eine erste Komponentenstruktur	definieren	5	
und				
vorausgesetzt, ich kann	den Nutzen einer initialen Grobstruktur zur Verhinderung einer „Software-Favela“ (ungesteuertem Wildwuchs) für mich	begründen	2	
indem ich	Use Cases aus der Aufgabenstellung	ableite	4	UseCase
	das fachliches Datenmodell und das Glossar aus der Aufgabenstellung	ableite	4	FachGlDM
	durch Clustering Subkomponenten	bilde	3	Clust
	die Cluster in eine erste Komponentenstruktur	transformiere	3	Komp
und abschließend	mit Hilfe der 7-Fragen-Methode die Unklarheiten / Aspekte zur Weiterentwicklung meines Entwurfs	analysiere	5	7Fragen
so dass ich	einen Startpunkt für die weitere Entwurfsarbeit habe.			

1.1.1 Definition von Use Cases

	UseCase			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	Use Cases aus der Aufgabenstellung	ableiten	4	
und				
vorausgesetzt, ich kann	Anwendungsgebiete und Nutzen von Use-Case-Diagrammen	erklären	2	
	Use-Cases-Diagramme der entsprechenden Perspektive (Nutzer-/Entwickler-, Verhaltens-/Struktur-Perspektive)	zuordnen	2	UseCase.Persp
indem ich	eine Übersicht über alle Use Cases, basierend auf meinem Anforderungstext, identifiziere und als Use-Case-Diagramm(e)	spezifiziere	3	UseCaseÜber
und abschließend	für jedes Diagramm ein entsprechendes Use-Case-Template	ausfülle	3	UseCaseTempl
so dass ich	einen Anhaltspunkt für die funktionale (verhaltensorientierte) Dekomposition meines Systems habe.			

1.1.1.1 Use-Case-Übersicht

	UseCaseÜber			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	eine Übersicht über alle Use Cases, basierend auf meinem Anforderungstext,	identifizieren	4	
und	als Use-Case-Diagramm(e	spezifizieren	3	
vorausgesetzt, ich kann	mich an Namenskonventionen für Use Cases	erinnern	1	
indem ich	den Anforderungstext in Blöcke von zusammenhängenden Aktivitäten	gruppiere	4	
	pro Aktivitätsblock eine treffende Formulierung	finde	3	
	die Aktoren des Use Cases	festlege	4	
und abschließend	das entsprechende Use-Case-Diagramm	zeichne	3	UMLUseCase
so dass ich	eine erste Übersicht der Anwendungsfälle (Nutzerperspektive) meines Systems vorliegen habe.			

1.1.1.1.1 UML-Use-Case-Diagramm

	UMLUseCase			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	funktionale Anforderungen an ein Softwaresystem in Form von UML-Use-Case-Diagrammen	modellieren	3	
und				
vorausgesetzt, ich kann	allgemeine Syntaxregeln für Use-Diagramme	erklären	2	
	die drei verschiedenen Beziehungstypen zwischen Use Cases	benennen	2	UMLUseCase.Bez
	die verschiedenen Typen von Aktoren (Menschen, Systeme)	erklären	2	
	die Rolle der Systemgrenze in UML-Diagrammen	beschreiben	2	
indem ich	die funktionalen Aktivitäten als Use Cases in meinem UML-Use-Case-Diagramm	modelliere	3	
	die Nutzer und Umsysteme als Aktoren	hinzufüge	3	
	die Beziehungen zwischen Use Cases passend	festlege	3	
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	die funktionale Sicht auf mein System in kompakter und leicht erklärbarer Form abgebildet habe.			

1.1.1.2 Use-Case-Template

	UseCaseTempl		
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler		
kann ich	die wesentlichen Eckpunkte des Anwendungsfalls detailliert	beschreiben	4
und	das Haupt-, Alternativ- und Ausnahmeszenario detailliert	spezifizieren	4
vorausgesetzt, ich kann	die Elemente eines Use-Case-Templates inhaltlich	einordnen	2
	mir den Unterschied zwischen „Nutzer“, „externes System“ und „System“ als Quelle von Aktionen	erklären	2
	die Regeln für syntaktische und semantische Bereinigung des Ausgangstextes für die Verwendung in Szenarienschritten	aufzählen	1
	mich an die Konvention für die Nummerierung von Ausnahme- und Alternativszenarien	erinnern	1
	die Bedeutung von Haupt-, Alternativ- und Ausnahmeszenario	erklären	2
indem ich	Randbedingungen wie auslösender Aktor, Auslöser, Vorbedingung und Nachbedingung	festlege	3
	Kandidaten für Aktivitäten (Verben) im Ausgangstext	markiere	3
	die Regeln für syntaktische und semantische Bereinigung des Ausgangstextes	anwende	3
	die Aktivitäten in einer Ablaufreihenfolge	anordne	3
	die Aktivitäten jeweils einer Swimlane (Nutzer, System, externes System)	zuordne	3
	die Aktivitäten entsprechend in das Hauptszenario des Use-Case-Templates	eintrage	3
und abschließend	das Alternativ- und Ausnahmeszenario analog zum Hauptszenario	spezifiziere	3
so dass ich	die detaillierte Abfolge der Nutzer- und Systemaktivitäten in meinem Anwendungsfall kenne.		

1.1.2 Erstellen von fachlichem Glossar und Datenmodell

	FachGlDM			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	die Geschäftsobjekte / fachlich relevanten Objekte	dokumentieren	4	
und				
vorausgesetzt, ich kann	den grundsätzlichen Nutzen eines Verzeichnisses fachlicher (geschäftlicher) Objekte	verstehen	2	
	den Zusammenhang zwischen Glossar und Datenmodell	erklären	2	
indem ich	das fachliche Glossar aus meinem Anforderungstext	ermittle	4	FachGL
und abschließend	aus den Elementen des fachlichen Glossars das fachliche Datenmodell als vereinfachtes Klassendiagramm	erstelle	3	FachDM
so dass ich	einen Anhaltspunkt für die strukturelle Dekomposition meines Systems habe.			

1.1.2.1 Fachliches Glossar

	FachGL		
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler		
kann ich	das fachliche Glossar aus meinem Anforderungstext	ermitteln	4
und			
vorausgesetzt, ich kann	die Regeln für syntaktische und semantische Bereinigung des Ausgangstextes für die Verwendung im fachlichen Glossar	aufzählen	1
	mir den Unterschied zwischen Geschäftsobjekt und -attribut	erklären	2
	Regeln für das fachliches Glossar als Tabelle	nennen	1
indem ich	(Teil-)Sätze zu Zielen oder Kontext der Applikation	streiche	3
	alle Hauptwörter	markiere	3
	die Regeln der syntaktischen und semantischen Bereinigung	anwende	3
	die Geschäftsobjekte und -attribute in eine Glossartabelle	eintrage	3
	aus dem Kontext und anderen Quellen stammende Geschäftsobjekte oder Attribute, die sich nicht unmittelbar aus dem Text ergeben,	ergänze	4
und abschließend	pro Geschäftsobjekt und -attribut eine kurze Erklärung, zusammengefasst aus meinen Quellen,	schreibe	4
so dass ich	eine Wissensbasis habe, um die Sprache meiner Fachdomäne korrekt zu verwenden und mit Fachexperten kommunizieren zu können.		

1.1.2.2 Fachliches Datenmodell

	FachDM			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	aus den Elementen des fachlichen Glossars das fachliche Datenmodell als vereinfachtes Klassendiagramm	erstellen	4	
und				
vorausgesetzt, ich kann	mir den Anwendungszweck von und den Unterschied zwischen fachlichem, logischen und physischem Datenmodell	erklären	2	
	diejenigen Elemente eines UML-Klassendiagramms, die im fachlichen Datenmodell vorkommen,	benennen	1	
indem ich	die Beziehungen zwischen den Geschäftsobjekten des fachlichen gemäß Textlage und meinem allgemeinen Domänenverständnis	analysiere	4	
und abschließend	ausgehend von den Geschäftsobjekten und -attributen sowie meiner Beziehungsanalyse das fachliche Datenmodell als vereinfachtes UML-Klassendiagramm	modelliere	3	UMLKlass
so dass ich	ein Verständnis der Geschäftsobjekte und ihrer Relation zueinander in der betrachteten Fachdomäne bekomme.			

1.1.2.2.1 UML-Klassendiagramm

	UMLKlass			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	eine textuelle Beschreibung (oder eine in anderer Form informell gegebene Domänenbeschreibung) in ein UML-Klassendiagramm	umsetzen	3	
und				
vorausgesetzt, ich kann	mir das Wesen eines Objekt, einer Klasse sowie den Unterschied zwischen beiden	erklären	2	
	die wesentlichen Bestandteile einer UML-Klasse (Attribute und Operationen)	zusammenfassen	2	
	mich an die UML-Notation für Klassen	erinnern	1	
	mir die Bedeutung von Assoziationen (als gemeinsame Struktur einer Menge von statischen Beziehungen zwischen Objekten)	erklären	2	
	die wesentlichen Eigenschaften von Assoziationen (Name, Rollen, Leserichtung, Multiplizitäten)	beschreiben	2	
	den Sinn, Optionen sowie Notation von Multiplizitäten (Kardinalitäten) von Assoziationen	erklären	2	UMLKlass.Mult
	mir den Unterschied zwischen den verschiedenen wichtigen Beziehungsstypen (Assoziation, Aggregation, Komposition, Generalisierung) sowie deren Notation	herleiten	2	UMLKlass.BezTyp
	spezielle Beziehungstypen wie die n-wertige Assoziation	erklären	2	
indem ich	Gruppen gleichartiger Objekt als Klassen	modelliere	3	
	Beziehungen zwischen Objektgruppen als Assoziationen vom jeweils passenden Typ	spezifiziere	3	
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	die Struktur der gegebenen Domäne in formal korrekter und standardisierter Form vorliegen habe, und somit der Wissensaustausch mit anderen Beteiligten am Softwareentwicklungsprozess ermöglicht wird.			

1.1.3 Clustering von Use Cases und fachlichem Datenmodell

	Clust			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	durch Clustering von Use-Case-Szenarioschritten und fachlichem Datenmodell Subkomponenten	bilden	4	
und				
vorausgesetzt, ich kann	die fünf zentralen Architekturprinzipien als Leitmotive meines Entwurfsprozesses	erklären	2	Clust.5Prinz
	gängige Antipattern gegen die fünf zentralen Architekturprinzipien	identifizieren	2	Clust.5PrinzAntiPattern
	Namensregeln für die Benennung von Clustern / Komponenten	nennen	1	
indem ich	alle Geschäftsobjekte (Elemente des fachlichen Datenmodells) auf einer (realen oder virtuellen) Fläche beliebig	anordne	3	
	in gleicher Weise alle Schritte der Use-Case-Szenarien	anordne	3	
	UC-Schritte und Geschäftsobjekte zu gemäß der allgemeingültigen Entwurfsprinzipien zu "eng zusammengehörigen" Clustern	zusammenstelle	4	
	Clustergrenzen	einzeichne	3	
und abschließend	die Cluster einfach und passend	benenne	3	
so dass ich	einen deutlichen Hinweis auf einen sinnvollen Schnitt der Komponenten meines Softwaresystems erhalte.			

1.1.4 Entwicklung einer Komponentenstruktur aus dem Clusterdiagramm

	Komp			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	die Cluster aus dem Clusterdiagramm in eine erste Komponentenstruktur	transformieren	3	
und				
vorausgesetzt, ich kann	die Begriffe Subsystem, Modul und Komponente	erklären	2	
	die Regeln für Beziehungen zwischen Komponenten (und deren Umsetzung mit Hilfe von Ports)	benennen	2	Komp.PortRegeln
	das Blackbox- / Whitebox-Paradigma zur schrittweisen Detaillierung durch Schachtelung von (UML-)Diagrammen			
	die drei möglichen Aufrufarten (synchron, asynchron, Pub/Sub bzw. Event)	beschreiben	2	Komp.AufrufArt
indem ich	zunächst die äußeren Cluster als Subsysteme in einem Top-Level-Komponentendiagramm	modelliere	3	UMLKomp
	Datenflüsse (basierend auf Geschäftsobjekten und UC-Szenarioschritten) als Aufrufbeziehungen zwischen Ports	spezifiziere	4	
	Ports gemäß der Datenflüsse aus meinen Informationsquellen sprechend	benenne	3	
	eingeschachtelte Cluster als Subkomponenten	modelliere	3	UMLKomp
	für die User-Interaktion UI-Komponenten	vorsehe	3	
	erste grundlegende Modellierungsentscheidungen, wie etwa das Herausziehen von Querschnittsdiensten als Umsysteme,	treffe	3	
	Umsysteme durch Farben und/oder Stereotypen	kennzeiche	3	
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	ein Bild meiner Struktur-/Entwicklungsperspektive (in Form einer Subsystem- und Komponentenstruktur) habe, die tragfähig genug für den weiteren Entwurfs- und Entwicklungsprozess ist.			

1.1.4.1 UML-KomponentenDiagramm

	UMLKomp		
Als	Softwarearchitekt oder Softwareentwickler		
kann ich	die Module meines Softwaresystems als Komponenten mit Schnittstellen und Aufruf-Abhängigkeiten	modellieren	3
und			
vorausgesetzt, ich kann	die wesentlichen Bestandteile einer UML-Komponente (und deren Notation)	beschreiben	2
	mich an die Tatsache, dass die tatsächliche UML-Syntax an dieser Stelle deutlich komplexer ist als in der Praxis meistens benötigt,	erinnern	1
	bei den Beziehungen zwischen Ports zwischen der Richtung des Datenfluss und der Aufruf-Abhängigkeit	unterscheiden	2
indem ich	Software-Module in Form von UML-Komponenten auf dem Diagramm	anordne	3
	Aufrufbeziehungen zwischen Komponenten	spezifiziere	3
	Ports als aufrufende und gerufene Schnittstellen	modelliere	3
	bei Schachtelung von Modulen entsprechende Subkomponenten	modelliere	3
	Implementierungsbeziehungen für gerufene Ports durch eingeschachtelte Subkomponenten	spezifiziere	3
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4
so dass ich	meine Komponentenstruktur in einer allgemeinverständlichen und standardisierten (und dadurch leicht erklärbaren) Weise spezifiziert habe.		

1.1.5 7-Fragen-Methode

	7Fragen		
Als	Softwarearchitekt oder Softwareentwickler		
kann ich	durch die Sammlung von einer begrenzten Menge von offenen Fragen (ca. 7) im Brainstorming-Modus die Aspekte zur weiteren Entwurfsarbeit	bestimmen	5
und			
vorausgesetzt, ich kann	mir den Sinn und Nutzen dieser Methode (=> ein Entwurf ist ein iterativer Prozess, bei dem Entscheidungen auch häufig nochmals revidiert werden müssen)	erklären	2
	mir die vier Entwurfsperspektiven (Nutzer-, Entwicklung-, Verhalten-, Struktur-) inhaltlich	erklären	2
indem ich	ca. sieben Fragen, die mich bei dem meinem jetzigen Entwurf verwirren oder stören,	aufschreibe	4
	diese informell in das Vier-Sichten-Raster	eintrage	3
	anhand der Eintragungen im Vier-Sichten-Raster die „Abdeckungsvollständigkeit“ meiner Fragen	prüfe	5
und abschließend	diese nach Dringlichkeit	priorisiere	5
so dass ich	eine Art von „Leitfaden“ für das weitere Vorgehen in meiner Entwurfsarbeit erhalte.		

1.2 Geschäftsprozesse und Rückschlüsse auf das Komponenten-Design

	GPKomp			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	die Geschäftsprozesse meines Systems	analysieren	4	
und	daraus abgeleitet mein fachliches Datenmodell und mein Komponentenmodell	anpassen	3	
vorausgesetzt, ich kann	die prägenden Eigenschaften eines (Geschäfts-)Prozesses	beschreiben	2	
	den Unterschied zwischen Nutzerszenario und Geschäftsprozess	erklären	2	
	die Regeln, wie ich bei einem Nutzerszenario und bei einem Geschäftsprozess jeweils die passenden Partitionen meines Prozesses festlege	erklären	2	GPKomp.Part
	die Entsprechung der Konzepte “Data Store Node” und “Central Buffer Node“ in fachlichem Datenmodell und Komponentendiagramm	benennen	2	
indem ich	die passenden Partitionen meines Geschäftsprozesses	festlege	3	
	den Geschäftsprozess, ausgehend von den Szenarien meines Use Cases (und anderen Quellen), als Aktivitätsdiagramm (zunächst ohne Objektflüsse)	modelliere	3	UMLAkt
	meine Informationsquellen (Use Case, fachliches Datenmodell, sonstige Quellen) auf Objektflüsse und Datenspeicher hin	analysiere	4	
	die gefundenen Objektflüsse und Datenspeicher dem Aktivitätsdiagramm	hinzufüge	3	UMLAkt
	sinnvolle Unterteilungen in Unterdiagramme (falls das Diagramm zu groß wird)	definiere	3	UMLAkt
und abschließend	das fertige Prozessmodell auf Rückwirkungen zum fachlichen Datenmodell (z.B. übersehene Geschäftsobjekte) und Komponentenmodell (z.B. fehlende Subkomponenten)	analysiere	4	
so dass ich	die Verhaltensperspektive meines Systems in einer Weise definiert habe, die tragfähig genug für den weiteren Entwurfs- und Entwicklungsprozess ist und mir analytische Verknüpfungen mit anderen Teilen meines Gesamtmodells erlaubt.			

1.2.1 UML-Aktivitätsdiagramm

	GPKomp			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	Abläufe als UML-Aktivitätsdiagramm	modellieren	3	
und				
vorausgesetzt, ich kann	die Abgrenzung zwischen Aktivitäts- und Zustandsmodellierung (und deren jeweiligen Nutzen)	verdeutlichen	2	
	das grundsätzliche „Knoten + Übergänge“ - Paradigma von Aktivitätsmodellierung	erklären	2	
	die Syntax von Aktivitätsdiagrammen in allen relevanten Details	auflisten	1	
	die (den Petrinetzen entlehnte) Semantik von Aktivitätsdiagrammen	beschreiben	2	
	gängige Modellierungsfehler bei Aktivitätsdiagrammen anhand von Beispielen	benennen	2	UMLAkt.ModFehl
	die Verwendung von Partitionen in UML-Aktivitätsdiagrammen	erklären	2	
	das Konzept von Datenflüssen (und die Abgrenzung von Kontrollflüssen) in Aktivitätsdiagrammen	beschreiben	2	
	die Konzepte “Data Store Node” und “Central Buffer Node“	erklären	2	
	die Konventionen, wie Verweise auf Unterdiagramme markiert werden	aufzählen	1	
indem ich	den zu modellierenden Vorgang als Kette von Aktionen und Übergängen (Kontrollfluss)	spezifiziere	3	
	mit Hilfe von Partitionen die Aktoren der Aktionen	verdeutliche	3	
	gemäß meiner Vorgaben und Informationen Objektflüsse und Datenspeicher	hinzufüge	3	
	das Aktivitätsdiagramm, falls es zu groß wird, um intuitiv verständlich zu sein, in ein Hauptdiagramm und hierarchisch geschachtelte Unterdiagramme	zerlege	3	
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	den Vorgang in einer allgemeinverständlichen und standardisierten (und dadurch leicht erklärbaren) Weise spezifiziert habe.			

1.3 Zustandsmodellierung von Geschäftsobjekten

	GOZust			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	die Zustände der Geschäftsobjekte in Form von UML-Zustandsdiagrammen	modellieren	4	
und	daraus abgeleitet mein fachliches Datenmodell und mein Komponentenmodell	anpassen	3	
vorausgesetzt, ich kann	das Wesen von Zustandsübergängen bei Geschäftsobjekten im Kontext eines IT-Systems	beschreiben	2	
	den Zusammenhang zwischen Aktivitäts- und Zustandsdiagramm	erklären	2	GOZust.AktZust
	Beispiele, bei denen die GO-Zustände besonders stark die (Verhaltens-)Semantik meines Systems prägen,	nennen	2	
indem ich	die Geschäftsobjekt Kandidaten für eine Zustandsmodellierung auf Basis von Anforderungen, Fachglossar und fachlichen Datenmodell	auswähle	4	
	die möglichen Zustände der Geschäftsobjekte gemäß der vorliegenden Quellen	analysiere	4	
	diese Zustände als Zustandsdiagramm(e)	modelliere	3	
und abschließend	das fertige Zustandsmodell auf Rückwirkungen zum Geschäftsprozessmodell (z.B. übersehene Aktivitäten) und Komponentenmodell (z.B. fehlende Subkomponenten)	analysiere	4	
so dass ich	der Verhaltensperspektive meines Systems weitere wesentliche Aspekte hinzugefügt habe.			

1.3.1 UML-Zustandsdiagramm

	UMLZust			
Als	Anforderungsanalyst, Softwarearchitekt oder Softwareentwickler			
kann ich	Objekt- und Systemzustände als UML-Zustandsdiagramm	modellieren	3	
und				
vorausgesetzt, ich kann	das grundsätzliche „Knoten + Übergänge“ - Paradigma Zustandsmodellierung	erklären	2	
	die Syntax von Zustandsdiagrammen in allen relevanten Details	erklären	2	UMLZust.Syntax
	inbesondere den Unterschied zwischen Wächtern (guards), externen Ereignissen (received events) und internen Ereignissen (actions)	beschreiben	2	
indem ich	die Zustände meines betrachteten Gegenstands als Knoten	anordne	3	
	diese durch passend beschriftete Übergänge und Kontrollknoten	verbinde	3	
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	die Zustandsfolgen und -übergange in einer allgemeinverständlichen und standardisierten (und dadurch leicht erklärbaren) Weise spezifiziert habe.			

1.4 Logisches Datenmodell

	LogDM			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	Datenhaltung und Datenzugriff durch die Weiterentwicklung des fachlichen zum logischen Datenmodell	definiere	4	
und				
vorausgesetzt, ich kann	den Zweck eines logischen Datenmodells sowie die Abgrenzung des logischen vom fachlichen und vom physischen Datenmodell	erklären	2	
	diejenigen Elemente eines UML-Klassendiagramms, die im üblicherweise im logischen Datenmodell vorkommen,	benennen	1	
	die häufigsten Antipattern bei der Umsetzung von Attributen und Beziehungen im logischen Datenmodell	erkennen	2	LogDM.AntiPattern
	gängige Formen der Optimierung des logischen DM an (Denormalisierung, „robuste“ Kardinalitäten)	aufzählen	2	
indem ich	in Schritt (1) die Liste meiner Geschäftsobjekte aus dem fachlichen DM durch Analyse der bisherigen Teilmodelle (Komponenten, Geschäftsprozesse, GO-Zustände)	vervollständige	4	LogDMGOVollst
	in Schritt (2) in analoger Weise die Attribute meiner Geschäftsobjekte auf Vollständigkeit hin (durch Vergleich mit den bisherigen Teilmodellen)	analysiere	4	
	in Schritt (3) die Beziehungen meiner Geschäftsobjekte auf Vollständigkeit	analysiere	4	LogDMBezVollst
	in Schritt (4) durch begründete Designentscheidungen die für die Implementierung irrelevanten Klassen und Beziehungen	entferne	4	LogDMIrrel
	in Schritt (5) Stamm-, Änderungs-, Bestands-, Bewegungsdaten	hinzufüge	4	LogDMDatArt
	in Schritt (6) das logische DM auf algorithmische Erfordernisse	optimiere	3	LogDMOpt
	alle Änderungen aus den Schritten (1) bis (6), basierend auf dem ursprünglichen fachlichen DM, begleitend in UML	modelliere	3	UMLKlass
	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	3	
und abschließend	in Schritt (7) die Bounded Contexts der Subsysteme / Komponenten bezüglich des logischen DM	analysiere	4	LogDMBC
so dass ich	eine Spezifikation der Datenhaltung meines Systems mit Blick auf Kommunikation der Komponenten und optimiert an funktionale und nichtfunktionale Anforderungen erhalte, die als Blaupause für die nachfolgende Implementierung dient.			

1.4.1 Stamm-, Änderungs- Bestands-, Bewegungsdaten

	LogDMDatArt			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	mein logisches Datenmodell in Bezug auf Stamm-, Änderungs-, Bestands- und Bewegungsdaten	analysieren	4	
und	diese entsprechend	modellieren	3	
vorausgesetzt, ich kann	die Natur von Stamm-, Änderungs- Bestands- und Bewegungsdaten (sowie deren Unterschiede)	erklären	2	
	die Modellierungsregeln für den Umgang mit diesen Datenarten	benennen	1	
	die häufigsten Antipattern im logischen DM bezüglich Stamm-, Änderungs-, Bestands- und Bewegungsdaten	erkennen	2	LogDMDatArt. AntiPattern
indem ich	Stamm-, Änderungs-, Bestands- und Bewegungsdaten in meinem logischen DM	identifiziere	4	
	Änderungs- und Bewegungsdaten durch Zwischenklassen (statt einfacher Assoziation)	expliziere	3	
	bei Stammdaten DB-unabhängige Immutable IDs	vorsehe	3	
	die obigen Änderungen begleitend in UML	modelliere	3	UMLKlass
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	über ein logisches Datenmodell verfüge, das auch transaktionale Vorgänge angemessen abbildet.			

1.4.2 Identifikation von Bounded Contexts im logischen DM

	LogDMBC			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	die Bounded Contexts der Subsysteme / Komponenten bezüglich des logischen DM	analysieren	5	
und				
vorausgesetzt, ich kann	das grundlegende Problem der "Verteilung" von systemweit (oder gar unternehmensweit) genutzten Daten auf Subsysteme	beschreiben	2	
	das Konzept des Bounded Context (BC)	zusammenfassen	2	LogDMBC.Konz
	die fünf wesentlichen Abgrenzungstypen bei überlappenden BC, samt ihren grundlegenden Vor- und Nachteilen,	erklären	2	
indem ich	meine Komponentenstruktur auf die Fälle von Überlappungen in der Datennutzung und -verwaltung hin	analysiere	4	
	diejenigen Subsysteme und/oder Komponenten, für die die Definition eines Bounded Context (BC) sinnvoll ist,	auswähle	4	
	in meinem logischen Datenmodell die BC der ausgewählten Komponenten	markiere	4	
	für alle überlappenden BC jeweils einen passenden Abgrenzungstyp	festlege	5	
und abschließend	meine getroffenen Designentscheidungen (z.B. in tabellarischer Form)	begründe	5	
so dass ich	eine Basis für das Schnittstellen- und Kommunikations-Design meiner Komponenten erhalte.			

1.5 Komponenten-Schnittstellen als Interfaces

	Schnitt			
Als				
kann ich	die Schnittstellen der Komponenten durch Interfaces	spezifizieren	4	
und				
vorausgesetzt, ich kann	das Konzept von Interfaces (in Abgrenzung zu Klassen)	erklären	2	
	die Syntax von Interfaces in UML-Klassen-Diagrammen	beschreiben	2	
	die Syntax von Interfaces in UML-Komponenten-Diagrammen (Ball-Socket-Notation)	erklären	2	
	die Kriterien, wann ein Port in Interface umgewandelt werden sollte,	zusammenfassen	2	Schnitt.KritInter
	die Beziehung zwischen Komponenten-Interfaces und dem logischen Datenmodell	erklären	2	Schnitt.LogDM
	die allgemeinen Grundsätze und den Nutzen des „Design by Contract“-Paradigmas	zusammenfassen	2	
indem ich	alle Ports in meinen Komponentendiagrammen auf die Notwendigkeit eines formal definierten Interfaces hin	analysiere	4	
	dort wo sinnvoll Interfaces mit Parametern aus dem logischen Datenmodell	spezifiziere	3	UMLKlass
	angebotene Ports im Komponentendiagramm durch Interfaces („Ball-Notation“)	ersetze	3	UMLKomp
	erwartete Ports im Komponentendiagramm durch erwartete Interfaces („Socket-Notation“)	ersetze	3	UMLKomp
	im Komponentendiagramm komplette Aufrufbeziehungen durch die Ball-Socket-Notation	ersetze	3	UMLKomp
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	die Kommunikation zwischen Komponenten durch einen formalen, extern bekannten Kontrakt geregelt habe.			

1.6 Datenaustausch der Komponenten durch Sequenzdiagramme

	KompSeq			
Als				
kann ich	den Datenaustausch der Komponenten durch Sequenzdiagramme	definiere	5	
und				
vorausgesetzt, ich kann	den Nutzen dieser Datenaustauschmodellierung im Kontext meines gesamten Entwurfsprozesses	zusammenfassen	2	
	die Quellen für die Erstellung des Sequenzdiagramms (GP-Modell als UML-Aktivitätsdiagramm, Komponentendiagramm, Interfaces)	beschreiben	2	
indem ich	aus meiner Geschäftsprozessmodellierung als UML-Aktivitätsdiagramm den als Aufrufsequenz zu modellierenden (Teil-)Prozess	auswähle	4	
	als Lebenslinien die beteiligten Komponenten aus dem UML-Komponentendiagramm	auswähle	4	
	die Interfaces, über der entsprechende Datenverkehr abzuwickeln ist,	bestimme	4	
	den Prozess(-ausschnitt) als Sequenzdiagramm (möglichst durchgehend unter Verwendung der Interfacemethoden)	modelliere	3	UMLSeq
und abschließend	die Komponenten-Interfaces bei fehlenden oder unpassenden Methoden	anpasse	5	
so dass ich	die bis hierhin existierende Komponenten- und Interfacemodellierung validieren und ggfs. überarbeiten kann.			

1.6.1 UML-Sequenzdiagramm

	UMLSeq			
Als	Softwarearchitekt oder Softwareentwickler			
kann ich	Aufrufsequenzen zwischen Instanzen von Klassen, Komponenten oder sonstigen Entitäten als UML-Sequenzdiagrammen	modellieren	3	
und				
vorausgesetzt, ich kann	die grundsätzlichen Bestandteile „Lebenslinie“, „Instanz“ und „Methodenaufruf“ eines Sequenzdiagramms	erklären	2	
	die Syntaxdetails dieses UML-Diagrammtyps	zusammenfassen	2	UMLSeq.Syntax
	insbesondere zwischen synchronem und asynchronem Aufruf	unterscheiden	2	
	die wichtigsten Interaktionsfragmente zur Abbildung von Varianten, Loops etc.	aufzählen	2	
	Alternativen zur Verwendung dieser Interaktionsfragmente, die das Diagramm übersichtlicher halten,	nennen	2	
indem ich	Instanzen als Lebenslinien in mein Diagramm	eintrage	3	
	die zu modellierenden Aufrufsequenz als Messages zwischen den Lebenslinien	eintrage	3	
	die Messages an konkrete Methodenaufrufe (falls solche in Form von Interface- oder Klassenmethoden vorliegen)	binde	3	
	Varianten, Loops etc. passend	modelliere	3	
und abschließend	eventuelle Besonderheiten meiner Modellierung in einem erklärenden Kommentar	festhalte	4	
so dass ich	die Aufrufsequenz in einer allgemeinverständlichen und standardisierten (und dadurch leicht erklärbaren) Weise spezifiziert habe.			