# POSA 1

## Layers

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Layering wird verwendet, um die Architektur in unterschiedliche Subtasks verschiedener Abstraktionsstufen zu unterteilen. | X |  |
| Ein Nachteil des Layers-Pattern ist es, dass aus der Änderung an einem Layer die Änderung aller Layer resultiert. |  | X |
| Sowohl die Unterteilung des Systems in Layers (logisch) als auch die Unterteilung in Tiers (physisch) sind ein Beispiel für das Layers-Pattern. | X |  |
| Layers sollten nur starke Abhängigkeiten vom abstrakteren Layer zum konkreteren Layer haben und nicht umgekehrt. | X |  |

## Pipes and Filters

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Buffering wird im Filter umgesetzt. |  | X |
| Buffering wird in der Pipe umgesetzt | X |  |
| Pipes and Filters macht die Fehlerbehandlung einfacher |  | X |
| Filter, Sources, Sinks und Pipes können gut wiederverwendet werden | X |  |
| Pipes and Filters erleichtert die Parallelisierung |  | X |
| Filter sollten immer parallel ausgeführt werden |  | X |
| Die Komponenten dieses Patterns sind: Pipe, Broker, Source, Sink, Filter |  | X |
| Die Filter dienen zur Übertragung der Daten |  | X |
| Der Data Sink dient als Datenquelle |  | X |
| Die Pipes sind für die Synchronisierung und das Buffering zuständig | X |  |

## Blackboard

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Blackboard-Pattern erlaubt eine einfache Austauschbarkeit der Knowledge-Sources, die voneinander entkoppelt sind und nicht untereinander kommunizieren | X |  |
| Die Knowledge-Sources können einfach ausgetauscht werden. | X |  |
| Aus verschiedenen Knowledge Sourcen kann bestimmt werden, welche die beste Lösung produziert. | X |  |
| Die Control-Komponente koordiniert die Knowledge-Sources und fordert diese ständig auf, auf dem Blackboard zu prüfen, ob sie etwas zur Problemstellung beitragen können | X |  |
| Chage Inspector ist eine Hauptkomponente des Blackboard Patterns |  | X |
| Die Resultate unterscheiden sich je nach Ausführung und sich nicht reproduzierbar. Dies führt zu schlechter Testbarkeit | X |  |
| Paralelitäts Probleme: Die Knowledge-Sources operieren alle auf einer gemeinsamen Datenstruktur (Blackboard), was dazu führt, dass die Zugriffe auf das Blackboard zwingend synchronisiert werden müssen. |  |  |
| Spracherkennung, Protein-Struktur-Identifikation, Kryptographie, Überwachung in der Luftfahrt sind Anwendungsbreiche für das Blackboard Pattern | X |  |
| Knowledge Sourcen werden in zwei Sub-Parts unterteilt: Condition-Part und Action-Part. | X |  |

## Model View Controller

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Der Controller erstellt die View, aber nicht das Model bei der Initialisierung | X |  |
| Hierarchical Event Handling kann durch eine Composite Architektur der Views erreicht werden | X |  |
| Das Model registriert abhängige Controller | X |  |
| Views und Controller stehen in einer 1:1-Beziehung zueinander. | X |  |
| Das MVC Pattern wird typischerweise in Applikationen mit UI eingesetzt | X |  |
| Im Controller werden die User Events in entsprechende Requests für das Model umgewandelt. | X |  |
| Das Observer Patterns wird für das MVC benötigt | X |  |
| Das Model teilt mit der Methode notify allen registrierten Komponenten mit, dass sich Daten geändert haben. | X |  |

## Presentation Abstraction Control

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Die Programmlogik des PAC Patterns findet sich in der Abstraction Komponente | X |  |
| Die Hierarchiestufen im PAC Pattern sind: Top-Level, Middle-Level, Bottom-Level |  | X |
| Unterschied PAC & MCV: Das PAC-Pattern teilt das System in hierarchische Komponenten auf, während das MVC auf eine einfache flache Gliederung setzt | X |  |
| Effizienz-Einbussen ist ein Nachteil beim PAC | X |  |
| Die Controle Komponente fungiert als Mediator | X |  |
| Daten des Repositories werden im Top-Level Agent geladen | X |  |
| Die Control Komponente ist zuständig für die Daten eines Agents. |  | X |
| Der Top-Level Agent fasst mehrere Bottom-Level Agents zusammen und stellt diese dar. |  | X |
| Intermediate-Level Agents sind zuständig für das Zustandsmanagement deren Bottom-Level Agents. |  | X |
| Bei grossen semantischen Konzepten welche die meisten ihre eigenen, einzigartigen Benutzerinterkationen haben bietet sich das PAC Pattern an. | X |  |

## Microkernel

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Moderne Operationssysteme nutzen das Microkernel Pattern |  | X |
| Microkernel uses kernel mode only. |  | X |
| Microkernel uses kernel and user mode. Monolithic kernel uses only kernel mode. | X |  |
| Eine Treibersoftware bei einem Betriebssystem ist ein Beispiel für einen Internen Server. | X |  |
| Eine Treibersoftware bei einem Betriebssystem ist ein Beispiel für einen Externen Server. |  | X |
| Interne Server verwalten die globalen Systemressourcen (wie bei einem Betriebssystem die File-Handles). |  | X |
| Microkernel verwalten die globalen Systemressourcen (wie bei einem Betriebssystem die File-Handles). | X |  |
| Der Client kommuniziert direkt mit dem Microkernel. |  | X |
| Der Client kommuniziert direkt mit den Internen Servern. |  | X |
| Der Client kommuniziert direkt mit den Externen Servern. |  | X |
| Der Client kommuniziert direkt mit den Adaptern. | X |  |
| Moderne Betriebssysteme basieren auch auf dem Microkernel-Pattern. | X |  |
| Ein Ziel des Microkernel-Patterns ist es, portable Applikationen schreiben zu können | X |  |
| Der Microkernel implementiert sämtliche Kernfunktionalitäten. |  | X |
| Die Internen Server implementieren sämtliche Kernfunktionalitäten. |  | X |
| Kernaufgaben des Microkernels: Bietet Kernmechanismen an, bietet Kommunikationsmöglichkeiten an, Kapselung der Systemabhängigkeiten, Verwalten und Kontrollieren von Ressourcen | X |  |
| Eines der Design-Ziele ist es, den Mikrokernel so klein wie möglich zu halten, um den Speicherbedarf zu reduzieren. Ein weiteres Ziel ist e s, Mechanismen zur Verfügung zu stellen, die schnell ausgeführt werden können für eine kurze execution time. | X |  |
| Die Kommunikation zwischen Client und external Server findet via sogenannte "Adapters" statt. Diese leiten die Requests zum Server weiter. Die Adapters bedienen sich für die Kommunikation der vom Microkernel zur Verfügung gestellten Infrastruktur. | X |  |
| Die Verwendung von Adaptern vrhindert eine starke Kopplung zwischen Clients und external Servers | X |  |

## Broker

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Ein Broker ist zuständig für: Weiterleitung von Client-Aufrufen an Server | X |  |
| Eine Bridge wird verwendet weil: Damit Broker und Broker miteinander kommunizieren können & Durch die Bridge ist die Technologie der Broker unabhängig | X |  |
| Bei einer direkten Kommunikation zwischen Client und Server ist der Broker zuständig für den Verbindungsaufbau | X |  |

## Whole Part

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| The Whole-Part is another Name for the Composite Pattern | X |  |
| Parts of the same Whole implement the same interface | X |  |
| Exchanging Parts of a whole changes the reference to the Whole |  | X |
| Delegation ist dann sinnvoll, wenn der "Part" eng in das "Whole" -Umfeld eingebettet werden soll | X |  |
| collection-members ist für die Gruppierung von ähnlichen Objekten geeignet | X |  |
| optimale Aufteilung der Parts ist von Anwendungsdomäne, modellierende Struktur, Funkionalitäten abhängig | X |  |

## Master Slave

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Ein Nachteil des Master-Slave-Patterns ist, dass es sehr schlecht für die Ausführung paralleler Aufgaben geeignet ist. |  | X |
| Die Berechnung der Fakultät einer riesigen Zahl kann durch verschiedene Slaves parallelisiert werden. | X |  |
| Ein System, welches Master/Slave einsetzt hat genau eine Master-Komponente und eine Slave- Komponente. |  | X |
| Master-Slave-Systeme können ungenaue Resultate erkennen und dienen deshalb der Rechengenauigkeit. | X |  |
| Slaves arbeiten immer parallel zueinander. |  | X |
| Es gibt immer nur genau eine Master-Komponente. | X |  |
| Master/Slave ist eine Variante des Prinzips "Divide And Conquer" | X |  |
| Die Anzahl Pfade beim "Problem des Handlungsreisenden" nehmen quadratisch mit den Anzahl Nodes zu. |  | X |
| Der Client kommuniziert direkt mit den Slaves. |  | X |
| Sub-Tasks werden durch den Master erstellt und via Parameter oder Repository (push/fetch) an die Slaves verteilt. | X |  |
| Fault Tolerance, Parallele Berechnungen, Genauigkeitsberechnungen sind Anwendungsbereiche für Master Slave Pattern | X |  |

## Proxy

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Ein typischer Anwendungsfall des Proxy-Patterns ist das Parallelisieren von verteilten Algorithmen. |  | X |
| Das Proxy-Pattern eignet sich sehr für das Cachen der Daten von Read-Only-Daten. | X |  |
| Ein Proxy eignet sich hervorragend zur Prüfung von Berechtigungen, bevor ein Vorgang durchgeführt wird. | X |  |
| Der Client ruft die Service-Funktion direkt von der Original-Komponente auf. |  | X |
| Der Client und Proxy implementieren das gleiche Interface. |  | X |
| Effizienz kann je nach Betrachtungspunkt des Proxy-Patterns ein Vorteil oder ein Nachteil sein. | X |  |
| Es gibt Fälle bei denen ein Proxy aufgerufen wird, ohne den Request auf das Original weiterzuleiten. | X |  |
| Ein Vorteil des Proxy-Patterns ist das Single Responsibility Principle | X |  |
| Die Verwendung eines Proxy Patterns steigert in jedem Fall die performance. |  | X |
| Der Zugriff auf Komponenten via Proxy sollte für den Client verborgen bleiben. |  | X |
| Die Verwendung eines Proxy kann die Effizienz steigern und kann auch aus Sicherheitsgründen praktisch sein. | X |  |

## View Handler

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Als gemeinsame View kann sowohl ein Interface wie auch eine Absrakte klasse definiert werden. | X |  |
| Observer Pattern wird oft bei den Suppliers für die Events angewendet | X |  |
| Die Komponente Specific View ist die zentrale Komponente |  | X |

## Forwarder Receiver

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Forwarder-Receiver-Pattern bringt den Vorteil, dass die Implementation der Interprozesskommunikation gut austauschbar ist. | X |  |
| Alle Peers laufen zwingend im gleichen Prozess auf der gleichen Maschine. |  | X |
| Alle Peers laufen zwingend auf der gleichen Maschine aber können auch in unterschiedlichen Prozessen laufen. |  | X |
| Peer-Komponenten können beliebig in verschiedenen Prozessen auf unterschiedlichen Maschinen laufen. | X |  |
| Der Receiver ist für die Serialisierung (Marshalling) der Nachrichten zuständig. |  | X |
| Der Receiver muss Funktionen für den Empfang und das Unmarshaling von IPC-Messages enthalten. Diese werden mit dem Blackboard-Pattern realisiert |  | X |
| Der Forwarder führt bei den Messages ein unmarshaling durch und der Receiver ein marshaling. |  | X |

## Client Dispatcher Server

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Im Client-Dispatcher-Server Pattern erstellt der Client direkt mit dem Server den Kommunikationskanal. |  | X |
| Der Server erstellt den Kommunikationskanal und registriert diesen beim Dispatcher |  | X |
| Der Dispatcher hält eine Map mit Servername und Location | X |  |
| Falls beim Client-Dispatcher-Service gleiche Services auf verschiedenen Server bestehen und der Verbindungsaufbau zum ersten Server nicht klappt, empfängt der Client immer eine Fehlermeldung |  | X |

# POSA 2

## Wrapper Facade

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Eine Wrapper Facade vereinfacht die Verwendung von low-level API Calls auf eine einheitliche Art und Weise | X |  |
| Eine Wrapper Facade versteckt betriebssystemspezifische Implementationsdetails vor dem Aufrufer | X |  |
| Eine Wrapper Facade versteckt komplexe objektorientierte Systeme vor ihrem Aufrufer |  | X |
| Das Wrapper Facade Pattern versteckt dahinterliegende Funktionen und kann als Blackbox angesehen werden. | X |  |
| Facade Pattern versteckt komplexe Klassenbeziehungen hinter einer einfachen API, während das Wrapper Facade Pattern kompexe Funktions- und Datenstrukturbeziehungen in grösseren APIs einkapselt | X |  |
| Error Handling: Beim Werfen einer Exception müssen die low-level Resourcen sauber aufgeräumt werden. | X |  |

## Interceptor

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Für die Event Notifizierung zwischen Framework<>Dispatcher<>Interceptor wird beim Interceptor Pattern das Observer Pattern verwendent | X |  |
| Interceptor Pattern löst folgende Probleme: Erweiterbarkeit, Service- Unabhängigkeit und Kontrolle über das Framework | X |  |
| Über Timeouts kann verhindert werden, dass bösartige oder fehlerhafte Interceptors das komplette System (konkretes Framework) blockieren | X |  |
| Bei Interception Kaskaden muss man mit Leistungseinbussen und Deadlocks rechnen | X |  |
| Je mehr Interceptor dispatchers ein konkretes Framework bietet, desto grösser das Risiko für Interception Kaskaden | X |  |

## Reactor

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Reactor Pattern nutzt mutli threading, um mehrere Requests gleichzeitig zu bearbeiten |  | X |
| Im Gegensatz zum Proactor Pattern werden beim Reactor Pattern die Events synchron verarbeitet. | X |  |

## Asynchronous Completion Token

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Beim Asynchronous Completion Token Pattern wird ein Token vom Service verändert, damit die Antwort des Services beim Aufrufer effizient verarbeitet werden kann. |  | X |
| Beim Asynchronous Completion Token Pattern ist dem Service der Kontext seines Aufrufers stets bekannt. |  | X |
| Beim Asynchronous Completion Token Pattern ist der Initiator für das Demultiplexieren zuständig. | X |  |

## Proactor

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Beim Proactor Pattern ist die Verarbeitung von Events strikt von asynchronen Mechanismen getrennt. | X |  |
| Ein Initiator kann auch gleichzeitig die Rolle eines concrete completion handlers übernehmen. | X |  |

## Acceptor Connector

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Beim Acceptor Connector Pattern kann ein Service immer nur entweder die Rolle des Servers oder des Clients einnehmen. |  | X |
| Der Connector ist für den passiven Verbindungsaufbau zuständig |  | X |
| Acceptors und Connectors sind nur für die Initialisierung von Service Handler und für den Verbindungsaufbau zuständig | X |  |

## Component Configurator

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Component Configurator Pattern lagert immer jede Komponente in einen eigenen Thread aus |  | X |
| Beim Component Configurator Pattern werden die Concrete Component statisch in die Applikation hineinkompiliert. |  | X |
| Stimmt es, dass es das Ziel des Component-Configurator Pattern ist, die Konfiguration einer Applikation dynamisch zu machen und zu entkoppeln | X |  |
| Das Component Configurator Pattern erlaubt eine klare Konfiguration des Verhaltens seiner Komponenten während der Laufzeit |  | X |
| Erlaubt das Component Configurator Pattern das Einfügen von Komponenten in verschiedene Anwendungsprozesse? | X |  |

## Active Object

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Beim Active Object Pattern läuft der Servant im gleichen Thread wie der Client |  | X |
| Beim Active Object Pattern läuft der Scheduler im gleichen Thread wie der Client |  | X |
| Beim Active Object Pattern läuft der Servant im gleichen Thread wie der Scheduler (im Thread des Active Object) | X |  |
| Beim Active Object Pattern läuft der Scheduler im gleichen Thread wie seine Proxies |  | X |
| Beim Active Object Pattern blockiert der Scheduler bis der Servant die Methode ausgeführt hat | X |  |
| Stimmt es, dass der Proxy und auch der Servant im Client Thread ablaufen? |  | X |
| Beim Active Object Pattern macht das Objekt aktiv gewisse Dinge, ohne dass es einen Input erhält. |  | X |

## Monitor Object

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Monitor Object des Monitor Object Patterns hält pro Aufrufer ein Monitor Lock. |  | X |
| Monitor Object Pattern: Wenn ein Thread, der das Monitor Lock gerade hält, blockiert, müssen alle Threads warten, bis er fertig ist oder auf die nächste condition variable wartet. | X |  |
| Beim Monitor Object Pattern kann ein Thread von sich aus das Monitor Lock freigeben, wenn er warten muss. | X |  |
| Beim Monitor Object Pattern wird der Monitor beim notify() direkt freigegeben. |  | X |
| Beim JAVA Monitor Object Pattern kann es zu einem Deadlock kommen, wenn mit geschachtelten Monitoren gearbeitet wird | X |  |

## Scoped Locking, Strategized Locking, Thread-safe Interface

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Scoped Locking Pattern kann in jeder Programmiersprache umgesetzt werden. |  | X |
| Beim Scoped Locking ist der Aufrufer zuständig, das Lock am Ende des Aufrufs wieder frei zu geben. |  | X |
| Scoped Locking ist ein generisch definiertes Pattern zur vereinfachung der Synchronisierung |  | X |
| Das Hauptziel von StrategizedLocking ist es, eine Synchronisierungsstrategie zu kapseln, so dass sie von allen Komponenten verwendet werden kann |  | X |
| Bei StrategizedLocking wird die Synchronisierung direkt in den Komponenten-Code programmiert |  | X |
| Beim Strategized Locking Pattern hat auf eine Komponente auf jeden Fall nur ein Thread gleichzeitig Zugriff |  | X |
| Das hier besprochene Pattern ermöglicht es Erweiterungen und Bugfixe für viele Komponenten auf einmal geltend zu machen, weil der Synchronisations-Mechanismus nicht direkt in den Komponenten programmiert ist. | X |  |
| Das Thread-Safe Interface Pattern verhindert Deadlocks |  | X |
| Das Thread-Safe Interface Pattern verhindert Self-Deadlocks | X |  |
| Beim Thread-Safe Interface Pattern wird die Performance durch ein Minimum an Synchronisations Direktiven erhöhat | X |  |

## Half Sync/Half Async

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Das Half-Sync/Half-Async Pattern benötigt Sychronisationsprimitiven im Queueing Layer, damit die beiden Layer kommunizieren können. |  | X |
| Das Half-Sync/Half-Async Pattern wird in 4 Layern umgesetzt: Async Layer, I/O Access Layer, Queueing Layer und Sync Layer. |  | X |
| Das GoF Pattern Mediator wird im POSA 2 Pattern Half-Sync/Half-Async verwendet | X |  |
| Synchronous Layer erhält direkt Daten/Nachrichten vom Asynchronous Layer. |  | X |
| Tasks im Synchronous task layer sind Passive Object welche keinen eigenen Runtime Stack oder Register besitzen. |  | X |
| Effizienz und Vereinfachung der Programmierung sind die hauptsächlichen Motivationen für das Pattern. | X |  |
| In der Implementation des Half-Sync/Half-Async Pattern sind in 4 Layer vorhanden. |  | X |

## Leader/Followers

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Im Leader Followers Pattern können Threads eine von drei Rollen spielen | X |  |
| Wenn im Leader Followers Pattern ein Thread einmal zu einem Follower geworden ist, kann er nicht mehr zum Leader werden. |  | X |
| Leader Followers Pattern: Der Leader wartet auf einen Event, wenn dieser auftritt delegiert der Leader die Arbeit an einen seiner Followers und wartet auf den nächsten Thread. |  | X |
| Ein gerade aus einem Prozess kommender Thread, kann direkt Leader werden wenn da keine andere Threads im Pool vorhanden sind? | X |  |
| Das hier besprochene Pattern mag zwar für high-performance Anwendungen geeignet sein, jedoch ist es nicht geeignet für gleichzeitig ablaufende System-Modelle welche multi Threading nutzen |  | X |

# Fault Tolerance

## Introduction

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Ein Fault wird oftmals auch "Bug" genannt | X |  |
| Konsistente Failures heissen deshalb so, weil Sie konsistent nur bei einem Benutzer des Systems auftreten |  | X |
| Ein Fault führt immer zu einem Error. |  | X |
| Können bei einem System ohne Spezifikation failures auftreten? |  | X |

## FaultTolerantMindset

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Quality != Fault Tolerance: Durch Qualität zwar besseren Code mit weniger Defects. Heisst aber noch lange nicht, dass der Code auch Fault Tolerant ist; siehe Abschnitt Quality vs. Fault Tolerance. | X |  |
| "Bei Hardware, welche in einer Telekommunikationszentrale zum Einsatz kommt, ist die Grösse der MTTR vernachlässigbar bzw. irrelevant." |  | X |

## IntroductionToThePatterns

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Soft Real-Time: Das Verpassen einer Deadline ist wohl ärgerlich, führt aber nicht zu einem katastrophalen Failure. Hard Real-Time: Das Verpassen einer Deadline führt zu einem katastrophalen Failure, daher werden viel strengere Anforderungen an ein solches System gestellt. | X |  |
| Ein Fault Tolerant System stellt meist eine grosse Investition dar (z.B. höhere Entwicklungs-Kosten). Entsprechend wird diese Investition eher geschützt und weiter entwickelt, anstatt ersetzt zu werden. Ein weiterer Grund gegen eine Neu-Anschaffung stellt die Verlässlichkeit des Systems dar. Das Fault Tolerant System wird daher über einen längeren Zeitraum eingesetzt und erhält dadurch den Charakter eines Long Lived Systems. | X |  |
| "Die fünf Phasen der Fehler-Toleranz beschreiben den Execution Time Lifecycle eines Faults." |  | X |

## UnitsOfMitigation Escalation CorrectingAudits

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| When an error occur, only this component is affected? |  | X |
| Is one advantage of correcting audits an enhanced availability? | X |  |
| Advantages of redundant system are: High availability, recover to an error-free state very quick, fault tolerant | X |  |
| Special rendunancy means to have copies of the system at different places (hardware). | X |  |
| Kann Design by Contract Bestandteil von Correcting Audits sein? | X |  |
| Escalation führt immer zu einem Neustart? |  | X |

## RecoveryBlocks

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Eine klare Verteilung der Verantwortung für das Error Handling führt zu einem Robusteren System. | X |  |
| Recovery Blocks sind ungefähr gleich komplex wie der Primary Block |  | X |
| die Akzeptanz-Tests stellen in diesem Kontext sicher, dass die Spezifikation eingehalten wurde |  | X |
| Verschiedene Implementationen für eine Aufgabenstellung ermöglichen Software Redundanz | X |  |
| Die Orte der Punkte selber, da aus denen die Distanz berechnet werden kann, wäre eine Sinnvolle Informelle Redundanz, wenn eine Distanz zwischen zwei Punkten gespeichert wird | X |  |

## MinimizeHumanIntervention

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is it okay if human operators do routine tasks because they have experience with the task? |  | X |
| Is it a good sign, if a system is quiet? |  | X |
| Macht es Sinn, auftretende Fehler direkt an eine zentrale Instanz (Someone in Charge) für die Fehlerbehandlung weiter zu leiten |  | X |
| das Ziel von Minimize Human Intervention: System kann Errors selbständig erkennen und beheben | X |  |

## MaximizeHumanParticipation

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the system ignore people totally? |  | X |
| Should mircowave oven user be able to override error processing? |  | X |
| Should the system ignore people totally? |  | X |
| Will errors reduced by people when they interact? | X |  |

## MaintenanceInterface

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should maintenance and application requests be intermingled on the application input and output channels? |  | X |
| benefit of providing a seperate interface for the maintenance: Only visible to those wo need it, save time while maintaining and therefore allows to make decisions more quickly | X |  |

## FaultObserver

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| The fault observer pattern is based on the publisher-subscriber pattern | X |  |
| Log Files can pile up and fill your space. You have to have a plan in that case. | X |  |
| separate the different observers in your system, So that the error doesn't get reported twice or more | X |  |
| Es können mehrere Observer im System eingesetzt werden (True),  jedoch dürfen sie nicht den gleichen Fault behandeln. (Falsch) |  | X |

## IntroductionDetectionPatterns

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Wenn ein Fehler durch die Applikation korrigiert werden kann, ohne dass das System in einen ungültigen Zustand gerät spricht man von Mitigatio n! | X |  |
| Error Detection, da ein Error noch innerhalb der Komponente ist und nicht - wie ein Failure - andere Komponenten in Mitleidenschaft zieht. | X |  |
| Units Of Mitigation sind eine Voraussetzung, damit ein Fehler isoliert werden kann | X |  |
| Bei der Fault Correlation (Fehler-Korrelation) geht es darum, herauszufinden, welcher Defekt in welcher Komponente einen Error ausgelöst hat. Nur so kann richtig auf den Fehler reagiert und am Ende auch der Defekt korrigiert werden. | X |  |

## FaultCorrelation

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Geht es bei der Fault Correlation um die Fehlerbehebung? |  | X |

## ErrorContainmentBarrier

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Error barriers keep errors in a unit of mitigation and try to solve problem in an isolated state. | X |  |
| when an error occurs, Mark the errornous data. Discuss ways of correcting it, do a rollback / fast-forward. | X |  |

## CompleteParameterChecking

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Are Assertions a good practice? |  | X |
| Could performance issues occure when you implement this pattern? | X |  |

## ExistingMetrics

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is introducing a new indicator for detecting overload a good idea? |  | X |
| pre-existing indicator on operating system: CPU idle time, RAM usage, ... | X |  |
| The systems indicators should be used in case of an overload | X |  |
| Is it a good idea to implement separate software which processes in case of an overload? |  | X |

## Voting

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Wird in beiden Voting-Arten (exakt und inexakt) jede Lösung von Redundanten Programmteilen mit vergangenen Lösungen verglichen? |  | X |

## SystemMonitor

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Fail-silent mode is the best failure mode for fault tolerant system, because it minimizes error propagation. | X |  |
| Is a monitor important in fault tolerant systems? | X |  |
| Is the location of a monitor important? | X |  |

## Acknowledgement

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Piggybacking means adding acknowledgement information to a reply. | X |  |
| It is possible to implement acknowledgement without dedicated messages only for acknowledgement. | X |  |
| Acknowlegement informs the requestor that another Task is still alive. | X |  |
| In a situation with limited network bandwidth acknowledgement is better suited than heartbeat. | X |  |

## Heartbeat

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| The System Monitor should initiate a request for heartbeat, if the monitored task doesn't know that it's being watched | X |  |
| Ist das Heartbeat pattern dazu da um Tasks am leben zu erhalten? |  | X |

## Watchdog

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| the watchdog is assigned to watch only one task whereas the system monitor watches a number of tasks. | X |  |
| Watchdog Doesn't add much complexity, doesn't change something in the monitored task. | X |  |
| The difference between a Watchdog and a System Monitor is the amount of observed tasks. | X |  |
| A watchdog can be a hardware or software component. | X |  |
| Watchdogs require explicit logic in the observed tasks. |  | X |
| A possible implementation of a watchdog, is measuring the time a task has taken for completion. | X |  |
| A component that is monitored by a watchdog does need to know about the watchdog. | X |  |
| A watchdog monitors multiple different tasks. |  | X |

## RidingOverTransients

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| If an error occurs more than once, it is not a transient error anymore. |  | X |
| A barometer in an airplane returns an unexpected value from time to time → This error needs processing | X |  |
| A connection has 50% packet loss → This error needs processing | X |  |
| A small percentage of web requests do not get answered regularly → This error needs processing |  | X |
| benefit of riding over transient: Higher availability and ressources are not wasted. | X |  |

## LeakyBucketCounter

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Gerät das System in einen ungültigen Zustand ist dies ein Beispiel für einen Fehler der "transient" ist. |  | X |

## RealisticThreshold

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| When creating realistic thresholds, the more critical a system is, the longer the Detection Latency should be. |  | X |
| A realistic threshold's messaging latency should be at least the maximal RTT. |  | X |
| A detection latency lower than the Messaging latency makes no sense. | X |  |
| A shorter Detection latency makes more sense when monitoring a critical task. | X |  |

## RoutineMaintenance

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| The routine maintence should be timed, that it executes when the system is very busy. |  | X |
| The routine maintenance is a good technique when you want to avoid many common and easily preventable errors. | X |  |
| Can data and memory allocations and routine checks allow detection of errors and failures earlier? | X |  |

## RoutineExercise

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should a routine exercise always be done? |  | X |
| The main point to use the routine exercise is when you have a redundant system. | X |  |
| benefit of routine exercise: Check for errors while there is not a huge workload active -> make systems more available. | X |  |

## RoutineAudits

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the data always be check at each access? |  | X |
| Should the data be examined for errors during idle time? | X |  |
| Latent Datenfehler müssen nicht an den Fault Observer gemeldet werden, sondern können direkt an Someone in Charge gemeldet werden, was zu eine r Escalation führt. |  | X |

## CheckSum

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is parity able to detect multiple errors? |  | X |
| Is it possible to correct errors (and not just detect them) by using checksums? | X |  |
| Are hashing algorithms able to compute checksums over large blocks of data? | X |  |
| In jeder Variante wird ein Parity Bit verwendet. |  | X |
| Checksummen sind hauptsächlich für integer values gedacht. |  | X |
| Hashing Algorithmen verwenden unter anderem Checksumme patterns. | X |  |

## IntroductionErrorRecoveryPatterns

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Error Recovery consists of detect the error and correct it |  | X |
| Error Recovery uses checkpoints, where a state is saved to be able to continue execution more quickly | X |  |
| The two types of error recovery are Processing error and return to error-free state, error mitigation to mask the error | X |  |
| the difficulty when using Checkpoints: Decide what to save, where to store it and when Checkpoints should be taken | X |  |

## CheckPoint

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| It is important for stateless systems to make periodic checkpoints. True or false? |  | X |
| Bei Grosser MTTF (also einer kleinen Fehlerrate) ist ein Checkpoint unnötig. | X |  |

## Restart

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the app be restarted if a small error occurs? |  | X |
| Should the app be restarted if there is an error in network? |  | X |
| Can a restart solve an error caused by a hardware fault? |  | X |
| Should a restart always be used as the last step in an U13\_3\_Escalation\_HS17 |  | X |

## RollBack

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the app be rolled back endlessly until there is no error? |  | X |
| Can the work contain side effects, if you want to roll back in case of error? |  | X |
| For realtime applications, redoing some of the work can be problematic? | X |  |
| Processing should resume at the point where a new work request is begun or at a checkpoint. | X |  |
| If a rollback is not succeeding, ever, manual shutdown is always the way to go. |  | X |
| Should the system take a rollback when requests cannot affored to be missed? | X |  |

## RollForward

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Rolling forward can be used in some time critical applications (voip). | X |  |
| Rolling forward ignores some requests | X |  |
| To apply this pattern, the results provided by the system have to independant from previous results, the result of the next request is the same independant whether the current request is ignored or not. | X |  |
| Should the system take a rollforward when requests cannot affored to be missed? |  | X |

## ReturnToReferencePoint

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is 'longjmp' an example of Return to a Reference Point? | X |  |
| Are reference points created during execution? |  | X |
| Sind REFERENCE POINTS dynamisch? |  | X |
| Bietet die C Funktion longjump eine Möglichkeit RETURN TO REFERENCE zum implementieren? | X |  |
| Referenz-Punkte sind dynamisch und werden zur Laufzeit festgelegt. |  | X |

## LimitRetries

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| If single problem-inducing transactions are not critical, they should be put into a QUARANTINE queue. |  | X |
| ROLL-FORWARD can be used for skipping failing system actions. | X |  |
| Gegen Denial of Service Attacken kann man ein System schützen, indem man seinen Zustand persistiert und nach einem Fehler wiederherstellt. | X |  |

## IntroductionErrorMitigationPatterns

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Ein System wird so gebaut dass genau einen Fault behandeln kann. | X |  |

## OverloadToolboxes

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is one toolbox enough? |  | X |

## DeferrableWork

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Tasks should not be deferred forever because that lowers the overall fault tolerance of the system | X |  |

## EquitableResourceAllocation

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
|  |  |  |

## IntroductionDetectionPatterns

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the requests be routed based only on U11\_3\_FreshWorkBeforeStale\_HS17 (aka LIFO)? |  | X |
| Should the requests be routed based only on priority? |  | X |
| LIFO queue not sufficient to ensure fair resource distribution, because Stale tasks are constantly overtaken by incoming tasks | X |  |

## QueueforResources

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the requests be discarded if system is overloaded? |  | X |
| Should the system have an infinitely long queue? |  | X |
| FIFO sollte verwendet werden bei Benutzeranfragen, um einen geregelten Ablauf zu garantieren. |  | X |

## MarkedData

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Für die Verwendung von markierten Daten müssen Regeln aufgestellt werden. | X |  |
| Es muss nichts mehr gemacht werden, wenn die Daten markiert sind |  | X |

## ErrorCorrectingCodes

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Can the regular checksum tell which part of data was modified? |  | X |

## ExpansiveAutomaticControls

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| To use the expansive automatic controls, the finite resource must be expandable | X |  |
| Expansive automatic controls blocks requests | X |  |
| The resources used in expansive automatic controls are always available to the system, the system can even use them during normal execution. |  | X |
| "Protective Automatic Controls" deals with a finite ressources. "Expansive Automatic Controls" deals with expandable ressources. | X |  |

## ProtectiveAutomaticControls

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| When a system is overloaded, the safest action is to not do anything, since it is performing well. |  | X |
| In aviation, FRESH WORK BEFORE STALE is used instead of QUEUE FOR RESOURCES because processing oudated messages may have deadly consequences. | X |  |

## ShedLoad

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is it best to shed work as early as possible? | X |  |
| One way of shedding work as early as possible is by using Shed Work at Periphery? | X |  |
| Does it help, accept all of the incoming work in an overflow? |  | X |

## FinalHandling

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should the resources of unexpected process termination be cleaned up? | X |  |
| Is maintenance costs more important than development costs (for fault tolerant systems) ? | X |  |
| Den Mechanismus zur Freigabe von Resourcen zu teilen, führt zu einer einfacheren Implementation des Mechanismus. |  | X |

## SlowItDown

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| When considering SLOW IT DOWN, the processing restrictions should be applied in predetermined steps. | X |  |
| SlowItDown sollte mit minimizeHUmanInteraction agieren. | X |  |

## ShedWorkAtPeriphery

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Why is it necessary to detect work for shedding as close to the edges of the system as possible?  Because then the work can be discard as soon as it arrives. | X |  |

## FinishWorkInProgress

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Finish work in progress means no new work is allowed |  | X |
| Finish work in progress means all work in progress must be finished, to accept new connections |  | X |

## FreshWorkBeforeStale

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Fresh work before stale uses LIFO | X |  |
| Fresh work before stale prioritizes incoming requests over older requests True | X |  |

## IntroductionTreatmentPatterns

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Fault treatment steps are, in following order, diagnosis, verification, correction |  | X |
| Fault treatment means repairing the fault or set is passive again | X |  |

## LetSleepingDogsLie

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Should you correct all faults when they are found? |  | X |
| Is the heartbleed fault in SSL a fault that should be corrected immediately? | X |  |
| Should every fault be corrected? |  | X |

## ReproducibleError

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is it possible for faults to be corrected or removed as side effects of other fault treatments? | X |  |
| Should faults always be corrected and never left alone? |  | X |
| Core of the pattern: Stimulate the fault in a controlled manner to verify that the fault did indeed cause the observed error and that the fault is still present in the system. | X |  |

## SmallPatches

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is small patches always the right choice? |  | X |

## RootCauseAnalysis

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| When performing a root cause analysis, asking 'why?' once is usually enough. |  | X |

## ReviseProcedure

|  |  |  |
| --- | --- | --- |
| Questions | True | False |
| Is a fault tolerance system normally maintained by a true expert? |  | X |
| Is it possible that an even correct procedure documentation can cause a failure? | X |  |