

Prüfungsfragen
ASE VO

1. What is ASE:

- Dependable Software
- Large & Complex Software
- Extended Software Lifecycle

2. What is Requirement Engineering?

Process of defining/documenting and maintaining SW-Requirements:
Epics → Features → Stories

User Story: As WHO i want WHAT so that WHY
(Independent/Negotiable/Valuable/Estimable/Small/Testable)

3. Unterschied zwischen **Error, Fault und Failure** erklären, Beispiel für jedes

Error: Difference between computed/measured/observed value and the real „true“ value

Fault: Abnormal condition that can cause failure

Failure: System can not longer perform an operation correctly

Example:

- Programmer may program an error (logical error, wrong variable, etc.)
- Error may lead to fault, could but does not have to lead to failure (Bug occurs in Program)
- Fault may lead to Failure (Programm crashes)

4. Define **Dependable Software**, explain 3-4 properties

Dependable Software is a concept with following **Attributes**:

Availability: System is ready to run

Reliability: System is always ready to run correctly

Safety: System does not lead to bad consequences for user/environment

Confidentiality: Information is only accessible for authorized users

Integrity: System does not alter improperly

Maintainability: System is easily repairable and extendable

Means: Fault Prevention/Tolerance/Removal/Forecasting

Threats: Error/Fault/Failure

Fault Prevention: QA + good Software Design

Fault Tolerance: Error detection & System recovery, Error Handling: Roll-Back

Fault Removal: Corrective Maintenance

Fault Forecasting: Qualitative (Classify/Rank) & Quantitative (Probability model)

5. Erklären sie das **Proxy Pattern**?

Example: Image interface (display()), ProxyImage (holds a ReallImage), ReallImage (can be on other machine)

6. What's the **difference** between a **Design Pattern** and an **Idiom**? Is the **Interface Pattern** implemented in Java an Idiom? Explain the Interface Pattern in Java. Ist it an Idiom?

Design Pattern:

- Lösung eines Problems in gewissem Kontext auf (höherer) Klassenebene
- Allgemeine Lösung
- Programmiersprachen unabhängig

Idiom:

- Domänen-spezifische Lösung eines Problems
- Konkrete Implementation eines Problems
- Programmiersprachen abhängig

7. **State Pattern** anhand eines Beispiels erklären

Example: File (States: open, close, delete), Abstract-Class State

For every state, concrete StateClass compared to using switch statement in one class
OpenState(close), CloseState(open, delete), Delete(none)

8. Is **Factory Pattern** an Idiom or Design pattern? Erkläre es warum.

Design Pattern!, platform independent, pattern for design structure on class level

Don't use new-Operator directly, if you have multiple SubTypes outsource Creation of these classes into one Factory class. Code for Creation of Objects is subject of constant change. These changes then have to be made only in Factory class.

9. Was sind die **Nachteile von Design Patterns**?

Do not lead to direct code reuse; Complex in nature; they are not always simple, they are validated by experience and discussion

Not always used in correct context

Can emerge to Anti-Pattern if it causes more Problems than it solves

10. What is an **Antipattern**? Name 3 of them as well as 3 reasons why they emerge, Nennen Sie 4 Gründe, warum Antipatterns entstehen!

Reasons why they arise:

Code is written by humans - humans make mistakes because

- Hasty decisions (wrong decision because of stress)
- Impassivity (Unwilling to solve known problem)
- Ignorance (Lack of seeking understanding, just copy/paste some solution)
- Pride (Try to reinvent the wheel, instead of using commonly accepted Design Patterns)

Examples Management Anti-Patterns:

Analysis Paralysis:

Goal, trying to plan/design the perfect system → too much project planning without actually implementing/testing something, over-analyze problem

Solution: Incremental Development, Details of Problem will be learned during process

Death by Planning:

Spend more time on planning than on development, focus on costs rather than development

Solution: Continuously update the plan, Incremental Deployment of Project

Smoke And Mirrors:

Management plans beyond company's capabilities

Solution: know your capabilities

ViewGraph Engineering:
Documentation over actual Development
Solution: Mockups, Simple Prototypes

FireDrill:
Work or get Fired!
Management prevents the development by always defining new/different directions
Management makes unrealistic demands for SW-Delivery (Project should be delivered yesterday)
Solution: Divide Development Environment → internal Environment (focus on continuous progress), External Environment (deliver stable releases to customer/outside world)

Email Dangerous:
Email is the wrong communication medium for sensitive topics

Examples SW-Anti-Patterns:
The Blob:
Use OO-language but architecture is actually procedural (few classes with hundreds of methods/variables, the rest dumb-objects)

Lava Flow:
Code-Blocks are stuck in a system and are never removed because uncommented without explanation, undocumented methods.

Golden Hammer:
One applicable/known Technology solves all our problems. Don't care about other (maybe) better technologies

Spaghetti-Code:
System with no structure (minimal relation to other objects, no OO-programming)

Exception Anti Pattern:
Expose lower-level Exceptions to upper layers → „API bleeding“
Log and Throw Exception
Catch or Throw „Exception“
Catch but Ignore
Throw inside finally block

11. Warum ist **Automatisierung** wichtig? Was sollte automatisiert werden?

Wichtig weil Development Prozess enorm beschleunigen/verbessern kann. In der agilen Software-Entwicklung besonders wichtig da nach jedem Sprint/Iteration fertige Software deployed werden muss. „Deploy as often as possible“.
Build-Automatisierung, Deploy-Automatisierung, Test-Automatisierung

12. Explain **Software Aging**

Programs just like humans get old with time which can lead to problems.
Example: A program that stores files (caches files) to a DB. The longer the user uses this program the slower the DB accesses can run.

Solution: Take preventive measures during development. OR Upgrade software afterwards to prevent aging. For long time, this is only possible for limited time.

Causes for Software Aging:

- **Lack of movement:** Failure to upgrade product to meet changing needs
- **Ignorant Surgery:** Updates are often made by different people who designed the system initially. Thus they don't know exactly what the programmer intended which can lead to errors.

Problems during LifeCycle:

- **Inability to keep up:** When software ages it gets bigger and bigger with each update, because easiest way to add new feature —> add new code
The bigger the code gets the harder it will get in the future to update/change something (more code has to be changed)
==> Result, someday it will no longer be possible to update code quickly enough

- **Reduced Performance:** See Example

- **Decreasing Reliability:**

Each corrected error with update can cause more than one errors

Each time we update software to decrease failure rate, it often gets even worse.

13. Unterschiede zwischen **OSGi** und Jigsaw erklären + in welchem Szenario würde man OSGi einsetzen? Purpose of OSGi. What is OSGi? Main components of OSGi Framework, Was versteht man unter OSGi und woraus setzt es sich zusammen? Nennen Sie prominente Implementierungen von OSGi!

What is OSGi:

OSGi is a Dependency Management Tool that binds dependencies (modules) during RUN-TIME (compared to Maven/gradle that do it during BUILD-TIME)

OSGi is only a specification and specifies APIs though and leaves implementations open to others. It only provides a reference-implementation.

Main components:

„Bundle“ and „Service“: Bundles/services are software components in OSGi e.g a Module that can be used in your Projects. These bundles can be dynamically installed, activated, deactivated, updated and de-installed.

Main advantage: You can install/update/remove „Bundles“ to your Application during runtime. Possibility to run modular Applications in parallel in the same virtual Machine and administer these Modules during the whole life-cycle of the Application. Dependencies between bundles are automatically resolved.

Prominent example for OSGi-Implementation:

Eclipse, Plug-Ins for Eclipse are OSGi-Bundles

14. Difference **Cathedral and Bazaar** model, 1. Beispiel zu cathedral model, Beispiel zu jedem

Cathedral:

- Source code is available for everyone on each Release
 - Code developed between Releases is only restricted to an exclusive group of developers
- Example: Emacs, GCC

Bazaar:

- Code is developed over the internet in public
- Can be seen and even written by everyone (first: Linux, nowadays: GitHub)

15. 3 business models of **FOSS**, (def. by Bruce Perens)

- Sell services for your FOSS: Training, Documentation (Books)
- Sell Extensions of your FOSS: ProVersion
- Gives you publicity for other Software and Services
- Gives you career opportunities because of publicity
- Sponsorship and Donations

16. What is FOSS (def. Bruce Perens) Name and describe 4 properties of FOSS according to Bruce Parens

Free Open Source System, in General: Source Code must be freely available.

Free Redistribution: You can use a FOSS in your own Software for free and sell your software that uses this component.

Source Code: The FOSS must provide the source code to a compiled version. If the FOSS is not shipped with the source code there must be well-publicized means to obtain the source code for free

Derived Works: It is allowed to modify a FOSS and distribute it under the same license as the original (not more restrictions → Copyleft)

No Discrimination against Persons/Groups: The license must not discriminate any persons

No Discrimination against Field of Endeavor: Anyone can use the FOSS in any field. No matter if it is for instance a large business company or a genetic research institution.

*Copyleft: Geändertes Werk darf nicht mehr Nutzungseinschränkung haben als Original

17. Vorteil wenn man Einen Bug im FOSS gefunden hat

Man muss keinen Workaround in seinem eigenen Code einbauen.

Sondern kann den Bug direkt in der OSS Library fixen.

Vorteil 1.) Dein Code bleibt der gleiche (kein Workaround notwendig)

Vorteil 2.) Die Community erhält ebenfalls den Bugfix

18. Explain three ways to monetize in FOSS.

- Provide a free OSS and add commercial value on top of the free version (Pro-Version)
- Offer professional training for your OSS
- Embed OSS into hardware (like Android did it)
- Project consulting (Help other companies to save money by using your OSS)

19. Difference between **re-licensing** and **sub-licensing**.

Sub-Licensing: Allows adding your OWN code in any license you like

Re-Licensing: Allows to change the license of an existing (licensed) code ==> not allow in ALv2.0

20. Explain the differences (pro/cons) between **Build Time Modularization and Runtime Modularization**.

Build Time	Run Time (single VM)	Run Time (multi VM)
Multiple JARs possible	Multiple JARs required	Multiple JARs required
Result: Single Bundle	Result: Multiple Bundles	Result: Multiple Bundles
Update of JARs requires complete redeploy	Update of JARs requires only partial redeploy	Update of JARs requires only partial redeploy
Easy operation	Medium - complex operation	Highly complex operation
Java Example: Maven	Java Example: OSGi	

21. Difference between **Clustering and Load-Balancing**? Name an example where they interweave

Load Balancing:

- Continuously check which servers are up
- New Request: Send it to one server (by policy)
- New Request of User with same session: send it to that same server

Clustering:

Cluster = group of resources (servers) with same setup that are aware of each other.
 Replicate their states
 Usually also includes Load Balancing (interweave)

Load Balancing can also happen without clustering: Multiple independent servers with same setup but are unaware of each other.

22. **Configuration Management** und 2. Aspekte. , 5 reasons why to use configuration and version management in the software development process, Irgendwas zu Configuration Management

In order to execute a Software Application it is necessary to configure its execution Environment (i.e. a computer/server). To achieve this multiple things need to be configured:

- Build Configuration
- Product Configuration
- Server/DB Configuration
- OS configuration
- System configuration

Tools like Chef or Puppet help with Configuration Management. With chef for instance you define recipes in which you can declare all configurations of a development system programmatically in Ruby.
 „Works for me“ —> Vagrant ...

23. Was versteht man unter **Continuous Integration**? Nennen Sie ein CI-Tool!, Name 5 CI principles according to Marting Fowler, Name two setup approaches of CI. Name 4 tools used in CI

Continuous Integration: Principle to constantly merge current development work into the Mainline of a Project.

(Git: Commit dev-progress to master everyday)

Continuous Delivery: Continuously deliver code to staging environment for customer

Continuous Deployment: Automatically build and deploy code from SCM (Git). For example: Nightly

5 **principles according to Fowler:**

1. Maintain Code Repo
2. Automate build (Jenkins)
3. Automatically execute tests on every build
4. Every dev commits to mainline everyday
5. build every commit to mainline
6. Everyone can see latest deliverables
7. Automate deployment

CI tools: Jenkins, Apache Continuum, CruiseControl, Bamboo (Atlassian)

Setup Approaches of CI:

1.) Internal (local) Build Server

Example: Setup internal Build Server like Jenkins

2.) Cloud Based Build Server

Use external Resources in the Cloud

Example: Travis CI

24. What is **Collective Intelligence**

CI in general: Group Intelligence that comes from collaboration of many individuals

CI in Computer Science:

„Groups of individuals doing things collectively that seem intelligent.“

Nowadays possible through:

- Easy/cheap communication between people over the world via Internet
- Web Apps (Social Networks, Wikis) that allow people to collaborate and work together
- Social Networks gather collective knowledge via user-generated content

Examples of CI Systems:

- Social Networks (FB, Twitter)
- Crowdsourcing (Amazon Turk)
- Content Sharing (YT)
- Knowledge Creation (Wikipedia)

25. Explain three main challenges of **Collective Intelligence systems**.

- **Designing the „right“ system:** Analyze user needs and optimize potential features

- **Perpetual beta:** Because CI's constantly change there is never a final release
- **Foster active community and contribution:** Keep users active and contributing.
- (Scale-up system: Big data/ Cloud computing/ Gloabal SW-Development)

26. Three trade-offs for **Centralized and Decentralized Collective Intelligence systems**, each.

Centralized	Decentralized
Acts as ONE platform	Distributed across multiple providers
Operated by ONE provider	Does NOT mean: run Centralized CI on mult. servers
One central admin/dev/content creation	consists of independent CI systems that communicate
FB, Wikipedia	GNU Social, Diaspora

Tradeoffs

Centralized	Decentralized
+ Constant QoS for all	- QoS dependent of individual node
+ Single point of access	+ Multiple point of access
- Closed Source code	+ Source Code often Open Source
+ Single organization responsible	- Each node responsible
- Government organizations	+ Easier to operate on non-profit basis
+ effective information exchange	- Ineffective information exchange
- Single point of failure	+ Easy hosting of new nodes

27. **Localization and Internationalization.** key aspects

Internationalisierung:

sämtliche Maßnahmen Software so zu gestalten, dass diese möglichst einfach an andere Sprachen + Kulturen angepasst werden kann (z.B. Android: strings.xml)

Lokalisierung:

Darunter versteht man den eigentlichen Übersetzungsprozess. Daher das anpassen von Datenformatierungen (Texte, Zahlen, Datum, Währung, etc.). Ist für jede Sprache/Kultur separat notwendig.

28. **Scrum** and **KANBAN** skizze und difference.

beide Prozess-orientiert

Scrum

Ziel: Hohe Produktivität durch hohe Anpassungsfähigkeit

Rollen und Elemente:

Product Owner: erstellt gemeinsam mit Kunden Product Backlog (Liste ALLER

Anforderungen), Team soll frei von Störungen arbeiten können

Scrum Team: 5-10 Entwickler + Scrum Master (sorgt für gutes Arbeitsklima)

Scrum Sprint: Erstellung von Sprint Backlog (Teilmenge des Product Backlog)
Dauer ca. 2-4 Wochen, jeder Sprint endet mit fertigem ausführbarem Softwareteil

Daily Scrum: Jeder sagt was er bis dato geschafft hat bzw. heute schaffen will

Sprint Review Meeting: Erfahrungen besprechen, Verbesserungsvorschläge diskutieren!

Resultat jedes Sprints: fertige Funktionalität (Increment)!



Kanban

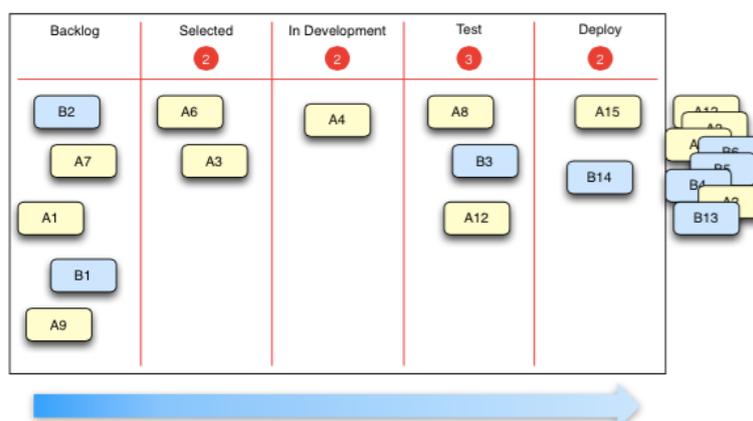
Ziel: WiP (Work in Progress) = Anzahl paralleler Arbeiten reduzieren → schnellere Durchlaufzeiten erreichen

Konzepte:

- **Arbeitsfluss** ständig **visualisieren**: Kanban-Board → Stationen werden als Spalten dargestellt
(Beispiel: Backlog | Selected | in Dev | Test | Deploy)

- Anzahl der Tickets (**WiP**) **limitieren** → Pull-System: Arbeit wird nicht einfach weitergegeben an nächste Station wenn fertig, sondern erst abgeholt wenn Stations-Limit erreicht wurde

- **Arbeitsfluss-Messungen** ständig durchführen → ständig sichtbar wie „gut“ Arbeit funktioniert
(Metriken: Bottleneck-Zeit, Zykluszeit, Durchsatz)



Unterschiede Scrum / Kanban:

Scrum	Kanban
fixe Iterationen (Sprint)	optionale Iterationen
Tickets müssen sich innerhalb Sprint erledigen lassen	keine Vorschrift bzgl. Ticketmenge
WiP wird durch Ticketmenge definiert	WiP wird limitiert
Aufwandsschätzungen	optionale Aufwandschätzungen
Tickets müssen priorisiert werden	keine Priorisierung
Rollenvorschreibung: Product Owner, Master, Team	keine Rollenvorschreibung
Board wird nach jedem Sprint neu aufgesetzt	Board wird kontinuierlich geführt

29. Two **transaction management possibilities** (pro/cons) in a Java EE Container.

Transaction: Operationen (meist DB-Operationen) die zwingend hintereinander (atomic) erfolgreich durchgeführt werden müssen. „Either all or none must complete.“

Möglichkeiten in JavaEE:

Container-Managed Transactions:

- Container übernimmt Transaction-Management automatisch.
- Man muss NICHT explizit Transaction starten und beenden sondern jede Methode (kann auch deaktiviert werden) für sich läuft in einer Transaction.
- Der Container beginnt eine Transaction bei Methoden-Aufruf und commitet die Transaction bevor die Methode beendet wird.

Bean-Managed Transactions:

- Developer muss sich selbst um Transaction-Management kümmern
- Daher: Explizit Transaction starten/commiten/rollback

Container	Bean
+ wenig Code	- viel Code
- less control	+ all control
+ easy to maintain	- difficult to maintain
- more complex when transactions go across methods Example: Stateful Bean	- can be complex and error-prone

30. Difference between **logging** and **auditing**.

Logging:

- Record implementation level events that happen as the program is running (methods get

called, objects are created, etc.).

- Use to detect bugs
- Understandable (mostly) only by programmers
- Amount of Detail can be configured at runtime

Auditing:

- Record domain-level events and tracing user activity (transaction created, user is performing action)
- Often legal obligation to record such events (banking-apps, medical-apps)

31. Name at least two common problems when keeping dependencies in a SCM?

- SCM's (especially Git) are not designed for managing binary files
- binaries cannot be „diffed“ ==> Changes cannot be seen
- no transitive dependencies
- no version information (except its included in filename)
- Updates of dependencies require deletion and re-adding dependency

32. What does a tool like **Nexus** manage?

Nexus OSS is a Repository Management tool.

- It allows to manage artifacts (like libraries, modules, etc.) and manage them as dependencies.
- Artifacts can be versioned and this way easily be found and updated.
- Also allows to differentiate between release/snapshot Repo's.
- With Nexus you can both, host your own repositories (like internal libraries) and also proxy/cache public repositories (like mavenCentral, jCenter(), etc.).

33. What is **Dependency Injection**?

Dependency Injection:

- Feature to achieve Inversion of Control
- Pattern where the gluing of objects is separated from implementation
- One central Container (Injector) that creates and binds objects and their relations
- Popular DI-Framework for JAVA: Spring

34. Was ist **NOSQL** und welche Vorteile hat es gegenüber SQL?

literally = Not only SQL

in practice = everything BUT SQL, So: just various alternative data store models (compared to relational principle)

Types of NoSQL DB's:

Key/Value —> just like a HashMap, Example: MemCached

Document Orientated: Extension on Key/Value, values are schema-free, each Key has a Document (JSON)

Columnar: Data stored in Tables but columnwise instead of row-wise

GraphDB: Data is stored in Graphs

Advantage to SQL:

- better scalability
- better performance

- many different types for different usages
- dynamic schema (schema free)

35. Was ist **SOA**? (mit dem SOA-Dreieck erklären)

Service Orientierte Architektur ist Architekturmuster um Dienste von verteilte IT-Systemen zu strukturieren.

- Orientierung an Geschäftsprozesse (Business Process Models = BPM)
- Bsp: „Vergib einen Kredit“, dahinter verbergen sich mehrere Geschäftsprozesse wie etwa „Eröffnen der Geschäftsbeziehung“, „Eröffnen eines oder mehrerer Konten“, „Kreditvertrag...“ und so weiter
- Anwendung werden durch mehrere Services (Orchestrierung) gelöst.
- Services sollen auch von anderen Anwendungen heraus ausgeführt werden können
- Charakteristika: lose Kopplung der Services, Standardisierung, Comosability (Zusammenfassung zu Komponenten)

36. Nennen Sie die Charakteristika von **REST** Services! Zeigen Sie den Zugriff auf eine Resource exemplarisch!

Represential State Transfer = Architektur-Pattern

HTTP 1.1 = Rest-Implementierung

Charakteristika:

- **Resourcen**: Jede Resource erhält URI (z.B: api/clients/)
- **Representations**: Ressourcen können verschiedene Datentypen haben (html, json, xml)
- **Operations**: 4 Standard Operationen auf Ressourcen (GET/POST/UPDATE/DELETE)
- **Hypermedia**: REST is designed for interaction, client changes application state using links
- **Statelessness**:

Server: Responsible for Resource-State

Client: Responsible for Application-State

Client-Server-Communication: Stateless

37. Lückenfüllen anhand angegebener missing words mit BOB and Alice. (Salomon,), Der Satz in den Folien wo Bob Alice mit LRDD findet und ihren Feed aboniert. (LRDD, PubSubHubbub, ...) waren in Lücken einzusetzen

Following a feed in a microblog:

—> Bob wants to subscribe to Alice’s public feed and Alice should be informed about it.

- 1.) Bob discovers the **LRDD** of Alice using **Webfinger**.
- 2.) Bob subscribes to Alice’s public feed using **PubSubHubbub**.
- 3.) Bob sends an **ActivityStream** Atom Entry to Alice using the **Salomon** protocol to notify her about the event.

LRDD = Link-based resource descriptor

Webfinger = Protocol to discover adressable entities

PubSubHubbub = Provide HTTP notifications for clients

Salomon = Message exchange protocol

JSON ActivityStream = Serialize Social Activity Streams in JSON

38. Warum ist für Facebook consistency nicht so wichtig? Erklärung anhand des **CAP-Theorems**, CAP Theorem erklären

CAP Problem existiert in replizierten verteilten Datenbanken .

Es können immer nur 2 der folgenden 3 Eigenschaften zur selben Zeit erzielt werden:

Consistency (gleiche Daten an allen Replikas) - **C**

Availability (alle Requests werden immer beantwortet) - **A**

Tolerance to **Partitioning** (System arbeitet auch bei Verlust einzelner Knoten) - **P**

Facebook: fällt unter **AP**

P - Last wird auf viele Knoten verteilt → wenn einer ausfällt muss System stabil bleiben

A - Availability ist Facebook oberstes Ziel → Server müssen immer und jederzeit verfügbar sein um User nicht zu vergraulen

C - Consistency nur zweitrangig: Wenn einzelne Nachrichten nicht bei allen Nutzern gleichzeitig eintreffen, ist dadurch die prinzipielle Funktion des Dienstes nicht beeinträchtigt.

AP: Beispiel DNS

- A → sehr hoch

- P → extrem hoch

- C X es dauert lange bis geänderter DNS-Eintrag an gesamte DNS-Hierarchie propagiert ist

CA: Beispiel RDBMS

- Consistency und Availability oberstes Ziel

- Werden aber oft mit hochverfügbaren Netzwerken und Servern betrieben → P niedrig

CP: Banking Anwendung

- Konsistenz extrem wichtig → abgehobener Betrag MUSS auf anderem Konto abgebucht werden

- Diese Anforderung muss auch bei Ausfall von Systemen gewährleistet werden

- Verfügbarkeit nur zweitrangig

39. Nennen Sie **4 Sensoren in einem Mobiltelefon** und geben Sie zu jedem einen konkreten Use Case an!

Beschleunigungssensor: Landscape/Portrait mode

Gyroscope: Gibt genaue Daten über Orientierung des Phones → Google Sky Map, zu welchem Sternbild zeigt dein Handy gerade

Lichtsensor: Helligkeit adjustieren

FingerprintSensor: Authentifizierung

Weitere/ältere Fragen aus dem Fragenkatalog

40. Was sind **Integration Styles**?
41. Was ist **Space-Based Computing** und Beispiele?
42. Was versteht man unter **Software Product Line Engineering**?

43. Erklären Sie die Vorteile von **Partitioned Iteration**!
44. Was ist der Vorteil eines **Dynamic Message Router** gegenüber eines Content-based Router? Verwenden Sie zur Erklärung ein Beispiel! Welche Probleme können beim Dynamic Message Router auftreten?
45. **Authorization** (Erklärung Role based access control and permission based)

46. Vorteile von Big Ball of Mud (seriously?!?)
47. Erklären Sie das **Chain Of Responsibility Pattern**!
48. Einfluss von **Cultural Diversity** auf distributed software engineering erklären + Gegenmaßnahmen
49. Drei Varianten von **Integration Patterns** erklären
?
50. In welchen Szenarien können drei der Softwarelebenszyklen (Build, Test, Code Checks, Report, Deploy) nicht automatisiert werden?
51. 4 Varianten Software für mobile Geräte zu implementieren erklären
52. Was unterscheidet Desktop- und Mobile-Applikationen bzgl. Development?
53. **Mining of software repositories** erklären
54. Welche Probleme treten beim **OR Mapping** auf?
55. Was sind die Vorteile von OO Datenbanken? Welche Limitierungen gibt es bei OO Datenbanken? Nennen Sie eine bekannte OO Datenbank!
56. Was sind **Enterprise Applications**? + 4 Charakteristika
57. Was ist der **Enterprise Service Bus**? + 2 bekannte Implementierungen
58. Vorbedingungen fürs **Refactoring**
59. Irgendwas mit **Falsification** und **Test Driven Development**
60. Gründe für unsichere Software (so in der Art)
61. What is **Model Driven Development**? Advantages and objectives? What properties does a model need in order to be used for MDD?
62. Concept of **NFC** + 1 Usecase
Estimate the time effort in % of the total project time for the phases planning, analysis, coding, design, testing and argue why you did so?
63. Explain **Context Aware Rules**
64. Model an example of a **Routine** (get up, turn on lights, make coffee)
65. What kind of quality checks shall be done at the closure of open issues?
66. Which human attitudes contribute to **Project Failure**?
67. Purpose of **Component Based Engineering**
68. Attributes of **complex system** + 2 examples