THEORY

This exam consists of 40 multiple choice questions. In every case only one of the answers is the correct one. You should indicate your answer in the corresponding separate answer sheet. All questions have the same value. If correctly answered, they contribute 0,25 points to the final grade. If incorrectly answered, the contribution is negative, equivalent to $1/5^{th}$ the correct value, which is -0,05 points. So, think carefully your answers.

The length of this part of the exam is 1 hour and 20 minutes.

1. Distributed systems...

A	...always consist of a set of concurrent agents that can be run in a set of interconnected computers.
B	...need to provide some degree of fault tolerance.
C	...allow resource sharing.
D	...may rely on message passing as their inter-agent communication mechanism.
E	All the above. All the items are true in this question. Multiple agents are needed. All they should be executed at once (i.e., concurrently). Multiple computers are needed to this end. This forces to have a network interconnecting all involved computers. Distributed systems need to provide failure transparency. This is equivalent to fault tolerance. Resources may be shared by the agents. The intercommunication mechanism may be message passing or memory sharing (usually, both mechanisms are used).
F	None of the above.

2. Some of the application areas in distributed computing are...

A	...location, replication, migration, persistency, transactional, access, and failure transparencies. Transparency is an objective of every distributed system. Thus, all these kinds of transparency are some of the goals to be achieved. None of them is an "application area". An application area is a given kind of computer applications that makes sense in the distributed systems field.
B	...WWW, sensor networks, *Internet of Things*, cooperative computing, highly-available clusters, cloud computing, etc. These are the areas presented in Unit 1 as some of the application areas of distributed computing.
C	...producer-consumer with bounded buffers, readers-writer problem, dining philosophers problem, etc. These are examples of classical concurrent programming problems.
D	...critical sections, distributed consensus, atomic broadcast, group membership, eventual consistency, 2-phase-commit protocol, nested transactions, etc. These are some general concepts, problems or mechanisms that may be found in the concurrent or distributed computing fields. None of them is an "application area".
E	All the above.
F	None of the above.

3. The generic scientific-technical goal of cloud computing is...

A	...to design centralised algorithms. Centralised algorithms usually introduce throughput bottlenecks. They are not recommended for building scalable distributed services. Cloud computing tries to provide support for scalable and adaptive distributed services. As a result, the design of centralised algorithms should be avoided, in the general case, in cloud computing.
B	...making money. This is not a scientific-technical goal but an economical one.
C	...to deploy fault-tolerant containers. A container is not a module that needs to be deployed, but the element being used for deploying components onto it. The main scientific-technical goal of cloud computing is not restricted to the deployment stage.
D	...to create and exploit software services in a simple and efficient way. This has been the main goal of cloud computing as explained in Unit 1.
E	All the above.
F	None of the above.

4. In the cloud computing area there are several roles related to the software service life cycle. Those roles are...

A	Web, worker and VM. These are the three types of roles/components in Windows Azure, but such roles are related to architecting a service in multiple components. This question is not about such kind of roles but on those related to the activities being needed in a software service life cycle process.
B	Monitoring, analysis, planning, execution and knowledge (MAPE-K). These are the five life cycle stages/elements being identified in the IBM specification of the autonomic computing approach. That was not the aim of this question.
C	User, developer, administrator and provider. These are the four roles being requested in this question. They have been explained in Unit 1.
D	SaaS, PaaS and IaaS. These are the three main cloud computing service models.
E	All the above.
F	None of the above.

5. What is the relation between concurrent and distributed systems?

A	Every distributed system is also a concurrent system. Distributed systems are composed by agents that collaborate in order to reach a common goal and are executed in multiple computers providing the image of a single and coherent system. If there are multiple agents and they collaborate, they are building a concurrent system. That is the definition of concurrent systems. Being a distributed system implies to be a concurrent system.
B	Concurrent systems are not distributed. Although some concurrent systems may be composed by agents being executed in a single computer (and this implies a non-distributed system), a subset of the concurrent systems contains all distributed systems; i.e., distributed systems are a subset of concurrent systems. So, this sentence is not always true.
C	Distributed systems are not concurrent. All distributed systems are concurrent. So, this sentence is clearly false.
D	Every concurrent system is also a distributed system. No. The correct implication is the reverse one: distributed systems are a subset of concurrent systems. For instance, a multi-threaded Java process that implements the dining philosophers problem consists of a set of shared resources (the forks) and a set of concurrent threads (one thread per philosopher). It is an example of concurrent application/system, but it doesn't need multiple computers or message exchanges. It is not distributed.
E	All the above.
F	None of the above.

6. In a distributed system, its agents may interact…

A	…using a message-passing mechanism. Yes. This is true. All the examples that we have seen in the ZeroMQ seminar show this fact.
B	…following a client-server approach. Yes. This is one of the existing approaches. Applications that use the ROI or RPC communication mechanism (or the REQ/REP communication pattern in ZeroMQ) follow this approach.
C	…following a peer-to-peer approach. This is also true. An alternative approach (to be used instead of the client-server interaction) is the peer-to-peer one. Specialised file distribution applications follow this interaction approach.
D	…sharing memory. True. Besides message-passing, agents may communicate sharing memory. When multiple agents are deployed into the same host, they may share memory in order to interact. Multi-threaded agents are an example of this kind.
E	All the above.
F	None of the above.

7. In a distributed system…

A	…each of its agents has a private state and does not interact with other agents. False. If it doesn't interact with the remaining agents, it is a separate and isolated process. It cannot be an agent of a given distributed application or system.
B	…agents may have their own state, but they collaborate in order to achieve a global goal. True. Each agent has its own state, although such state does not need to be private (some of its parts could be shared with other agents). Additionally, distributed system agents should collaborate. This is the essence of concurrent and distributed agents. Otherwise, they would be independent processes.
C	…agents are independent and they do not share resources. False. To be completely independent means that they do not build a concurrent system. Non-concurrent systems cannot be distributed systems. To share resources is one of the characteristics of distributed systems.
D	…concurrency is the source of many problems. So, modern distributed systems are not concurrent. False. Concurrency is inherent to distributed systems. Distributed systems need to be concurrent.
E	All the above.
F	None of the above.

8. The peer-to-peer interaction approach…

A	…is not used in distributed systems. False. It is one of the interaction approaches in distributed systems. Another one is the client-server approach.
B	…assumes that agents are interested in some kind of resource and as soon as some agent B has an instance or fragment of such resource, B may distribute it. True. This is the core of a peer-to-peer interaction approach. Once an agent gets a resource it may provide it to other agents. As a result, there is no distinction between client and server roles. All agents behave in both ways.
C	…clearly distinguishes between client agents and server agents. No. In the regular case, agents in a peer-to-peer interacting system are known as servents. This is the contraction of SERVer and cliENT.
D	…is a strongly centralised interaction approach. No. Peer-to-peer systems are decentralised in the regular case.
E	All the above.
F	None of the above.

9. The world-wide web…

A	…is an example of distributed application following the peer-to-peer interaction approach. No. The world-wide web application area regularly uses a client-server interaction approach.
B	…uses web browsers as a particular type of server agent. No. Web browsers commonly play a client role in client-server interactions.
C	…is a type of distributed application area where documents are transferred between servers and clients. Yes. In the regular case, documents (i.e., static web page contents) are transferred from servers to clients.
D	…does not allow a client-server interaction approach. No. It regularly uses that interaction approach.
E	All the above.
F	None of the above.

10. About the service models in cloud computing:

A	IaaS: Provides general applications as its service. An example is Google Docs / Google Drive. No. That is the definition and an example of SaaS.
B	SaaS: Automatizes application deployment and application elasticity. An example is Windows Azure. No. That is the definition and an example of PaaS.
C	PaaS: Provides a virtual infrastructure as its service, where components can be deployed in a non-automatized way. Example: Amazon EC2. No. That is the definition and an example of IaaS.
D	IaaS relies on the service provided by SaaS which also relies on the service provided by PaaS, defining a three-layered logical tree of service dependencies. No. SaaS is above PaaS and PaaS is above IaaS.
E	All the above.
F	None of the above.

11. The properties being required to distributed systems are...

A	Centralised control. No. Centralised control hardly scales out. Scalability is a requirement for regular distributed systems or applications.
B	Daily software upgrades. No. There is no need to upgrade the software so often. As in any other software area, upgrades are needed for fixing bugs or for extending the functionality of the software.
C	Extremely high degrees of concurrency in each implemented agent. Not necessarily. Distributed systems are inherently concurrent, but such concurrency may be obtained running multiple single-threaded agents or processes. For instance, Node.js components are single-threaded.
D	To be programmed in Node.js. Not necessarily. Distributed agents may be implemented in different programming languages. Java, C and C++ are valid examples. Distribution does not depend on the specific programming language being used.
E	All the above.
F	None of the above.

12. Some of the fundamental problems (and their solutions) in distributed computing are...

A	Component coordination (via message passing, designing algorithms that require a minimal message exchange).
B	Failure management (using replication, failure detectors and recovery mechanisms).
C	State persistence (via distributed commit protocols, persistent storage and replication).
D	State consistency (using replication and consistency protocols).
E	All the above. Unit 2 has presented coordination, failure management, state persistence and state consistency as four of the fundamental problems in distributed computing. Some of their possible solutions have been included in each of the previous items, in parentheses.
F	None of the above.

13. The distributed system model presented in Unit 2...

A	...considers all low-level details about system behaviour. This guarantees a more precise result in the software design stage. No. Low-level details are never considered in models. Models remove all such details and present an image of a given system or mechanism at a high level of abstraction.
B	...assumes that all agents are multi-threaded processes. No. The model assumes that agents are sequential processes that are represented as automata. The execution of an action is modelled as a state transition.
C	...always assumes synchronous processes and synchronous communication. False. For instance, communication may be asynchronous. Different degrees of synchrony had been defined, considering multiple aspects: communication, clocks, channels, processes,... Indeed, we suggested that communication needs to be asynchronous in order to boost scalability.
D	...represents the execution of processes as a sequence of interruptible actions or events. No. Actions should be atomic; i.e., uninterruptible. Events and actions are not synonyms. An event behaves as a guard or precondition for the execution of an action. This means that the action is not started until the event is raised.
E	All the above.
F	None of the above.

14. When we compare asynchronous servers with multi-threaded servers...

A	Asynchronous servers implement in a trivial way the atomic actions defined in the proposed distributed system model in Unit 2. True. Asynchronous servers are single-threaded and they use their events as guards or preconditions that should be set in order to start the execution of an action. Those actions are not interrupted once they have been started. So, they are run in an atomic way.
B	"Event-driven" is a synonym for "multi-threaded". No. "Event-driven" is a synonym for "asynchronous server".
C	Asynchronous servers are commonly blocked due to concurrency while multi-threaded servers tolerate concurrent accesses to resources without blocking. No. Each asynchronous server is single-threaded. So, they cannot be blocked due to concurrency. Indeed, they cannot be blocked at all. On the other hand, multi-threaded servers need some concurrency control mechanism for protecting their internal critical sections. This may lead to blocking some of their threads in some cases.
D	JavaScript is an example of programming language specifically tailored to implement multi-threaded servers. No. It is an example of programming language for implementing asynchronous servers.
E	All the above.
F	None of the above.

15. Properties required to distributed systems…

A	Fault-tolerance.
B	High availability.
C	Security.
D	Scalability.
E	All the above. True. These four properties have been listed in the introduction of Unit 2 as being required in every distributed system.
F	None of the above.

16. State consistency means that…

A	All the state being managed in a component may only have a single copy in all the system; e.g., it is stored in a centralised database. No. State consistency is a synonym for replica consistency. It makes sense when there are multiple copies of a given data element. This sentence states the contrary. It assumes that a single copy exists in all the system. In that case, there are no replicas. So, we cannot talk about state consistency or about the consistency model being used in such system.
B	All global variables should be accessed in mutual exclusion in order to avoid race conditions. No. The term "state consistency" in a distributed environment refers to the degree of divergence/consistency among the multiple copies of a given data element. Replication is regularly used in distributed systems. Race conditions may also generate inconsistencies, but such inconsistencies aren't related with divergences among multiple replicas in a distributed system. The model being proposed in Unit 2 recommends the usage of asynchronous (and single-threaded) servers. In that case, we don't need to worry about how the variables of a given process are accessed. Race conditions don't arise in that scenario.
C	When a component is replicated, there is a set of invariants that limit the degree of divergence among the replicas of a specific data element. True. This is the meaning that we have presented in Unit 2 for "state consistency".
D	When a component is replicated, either all replicas are alive and work correctly or all they fail and are unable to work. False. It makes no sense to compel all replicas to be stopped at once. Replication is used for guaranteeing service continuity. When a replicas fails, the remaining ones should hide such failure.
E	All the above.
F	None of the above.

17. State persistence means that…

A	A distributed application cannot have volatile data. All managed data elements should have a copy in disk files or databases. False. When state persistence is guaranteed, we ensure the durability of the data. However, this doesn't prevent us from using other temporary variables. For instance, in order to maintain some derived data elements that may be computed from the persistent data. This might be useful when such derived values are often needed. Instead of computing them each time, we may maintain them in non-persistent variables.
B	The access to any data element should be always done in the context of a distributed transaction. False. The term "distributed transaction" refers to a set of several subtransactions. Each subtransaction is executed in a different processor or computer. All those subtransactions collaborate in a global sequence of operations that ensures atomicity and isolation properties. It is not mandatory to access persistent data elements in the context of a distributed transaction. Local transactions are enough in most cases.
C	Once a persistent data change is applied, its endurance should be guaranteed. True. This is the meaning being used in Unit 2 for the "state persistence" concept.
D	Every secondary storage device being used by a distributed application should be replicated. False. Replication is not necessarily applied at the secondary storage "device" level. This would mean that all disks being used in the distributed system were RAID (Redundant Array of Inexpensive Disks) devices. This is not the common approach in distributed systems. Additionally, state persistence doesn't necessarily depend on secondary storage devices. Some highly-performant database management systems maintain their data in main memory, using multiple computers with large amounts of RAM. VoltDB is an example of this kind.
E	All the above.
F	None of the above.

18. In the distributed system model of Unit 2…

A	Internal events refer to actions applied by the logic of the agent. For instance, to receive a message. False. Internal events have such definition, but the reception of a message is not an internal event. It is an external event since it is related to an action that was initiated by another agent. It is external.
B	Internal and external events generate state transitions. True. Events (of any kind) generate state transitions since they start the execution of an action.
C	The execution of an agent is modelled as a sequence of events. So, it is always sequential and both multi-threading and concurrency cannot be represented. False. The first sentence is true, but the consequence being described in the second sentence is wrong. Although multi-threading is not directly supported by the model, this doesn't imply that concurrency is not allowed. A distributed system consists of multiple agents. Despite being each one a sequential agent, the overall picture is a clear scenario of concurrent executions. Those agents share some resources. They collaborate in order to achieve a common goal. Agents interact among them. As a result, they compose a concurrent system. So, concurrency may be represented in this simple system model.
D	Since all distributed systems should be failure transparent, this model assumes that failures never happen. False. Failures may happen. This simple system model assumes that agents follow a stop (i.e., the most benign) failure model.
E	All the above.
F	None of the above.

19. Communication in the simple system model of Unit 2…

A	…assumes that internal events define a "locally-precedes" total-order relation in each agent. True. This is the FIFO condition being considered in the "causal" relation.
B	…assumes that external events define a "directly-causes" relation where an output event is the cause of an input event. True. The typical output event is to send a message. The common input event is to receive a message. Message transmission is modelled by the "directly-causes" condition of the "causal" relation.
C	The transitive closure of the "locally-precedes" and "directly-causes" relations defines the "causal" communication relation. True. Lamport defined the causal relationship as the transitive closure of two elementary conditions (internal sequence of events, i.e., the "locally-precedes" condition, and communication-related events, i.e., the "directly-causes" condition). Those conditions received those names in Unit 2.
D	The "causal" communication relation allows to identify unrelated events as "concurrent". True. The "causal" (a.k.a. "happens before") relation sets causal dependences between events. When a pair of events are unrelated (i.e., we cannot say that "a → b" nor "b → a"), we may state that those two events are concurrent (i.e., "a || b").
E	All the above.
F	None of the above.

20. In order to specify programmes in the simple system model of Unit 2…

A	The model assumes atomic guards, protected by actions. False. The model assumes atomic actions (i.e., fragments of the programme that will be executed in a single and uninterruptible step; for instance, Node.js functions), protected by guards (that behave as preconditions for those actions).
B	Atomic actions are a potential source of errors. Because of this, they are implemented as interruptible blocks of code in all programming languages. False. Atomic actions avoid race conditions, so they are a building block that prevents many errors from appearing when a programme is run. Their atomicity property should be carefully maintained when that simple system model is translated into a real system. The programming language being used should preserve such atomicity. This means that each block of code that implements an action should be uninterruptible.
C	Guards are a potential source of race conditions. So, they are not used in multi-threaded programming languages. False. Guards and atomic actions prevent race conditions from appearing. So, they should be carefully translated to our chosen programming language. For instance, when such programming language is multi-threaded, atomic actions might be translated into monitor operations and guards could be the conditions being managed in those monitors. Such approach was carefully explained in CSD.
D	The model assumes atomic actions, protected by conditions (also known as guards). True. This has already been discussed in the first part of this question.
E	All the above.
F	None of the above.

21. Middleware is a software layer that...

A	...is placed between the hardware and the operating system. No. It is placed above the operating system, providing some degree of distribution transparency to the distributed applications developed on top of it.
B	...guarantees failure transparency to the components of distributed applications. No. We have presented different types of middleware in Unit 3. Many of them do not deal with failures nor provide component replication. As a result, they are unable to guarantee failure transparency. For instance, RPC mechanisms are an example of middleware. They may provide location transparency, but they do not automatically replicate any component. So, when a server fails, its client gets an exception as a result of its request.
C	...relies on containers for deploying distributed services. No. Many middleware layers do not deal with component deployment. So, they do not depend on any way from containers.
D	...is implemented in JavaScript. No. Middleware layers are not necessarily implemented in JavaScript. We are interested in the functionality, mechanisms and protocols being provided by a given middleware, but not in the programming language that was used in its implementation.
E	All the above.
F	None of the above.

22. Some characteristics of all middleware layers are...

A	They provide standard APIs.
B	They use standard interaction protocols.
C	They provide services of general interest.
D	They guarantee the interoperability of components deployed on distinct platforms.
E	All the above. All these characteristics were listed in the student guide for Unit 3. Those characteristics were also explained in the research paper from Phil Bernstein that presented and classified these software layers.
F	None of the above.

23. Distributed object systems...

A	...need a middleware for managing remote object invocation. Yes. An *object request broker (ORB)* is used to this end. It is a specific kind of middleware.
B	...are inherently less scalable than distributed systems based on message-oriented middleware. Yes. They use a client-server interaction approach. This implies a synchronous communication pattern (similar to the REQ/REP of ZeroMQ) that blocks the requester (or client) until an answer is sent by the server. Every source of blocking should be avoided in order to implement scalable distributed systems.
C	...have a higher coupling than non-object-oriented distributed systems. Yes. Distributed object systems use object references in order to access remote objects. In many cases, this implies that many data should be transferred between different system components in every object invocation. This demands large messages and reduces the system throughput. A high rate of exchanged messages between interacting components is an indicative of high coupling, and this should be avoided in scalable systems.
D	...are, in the common case, location-transparent. Yes. The use of proxies in the client side provides location transparency. The client process doesn't know on which computer the invoked object is placed.
E	All the above.
F	None of the above.

24. Message-oriented middleware...

A	...is persistent when the sender remains blocked waiting for some kind of reply from the receiver. No. It is synchronous in that case, instead of "persistent".
B	...is transient when the communication is managed by a broker agent. No. Brokers are only needed in persistent communication. Transient communication doesn't need any broker.
C	...may be persistent and broker-based. Yes. That is a valid combination. The AMQP (Advanced Message Queueing Protocol) standard follows such an approach.
D	...may be synchronous and transient. ZeroMQ is an example of this type. No. Such combination is valid and it exists. For instance, RPC is an example of such kind of message-oriented interaction. However, ZeroMQ isn't an example of that type. ZeroMQ is weakly persistent (instead of transient), brokerless, and asynchronous (instead of synchronous).
E	All the above.
F	None of the above.

25. Standards…

A	…make interoperability easier. True. This is one of their main targets.
B	…cannot be used in distributed services. False. Middleware layers implement standards and are used in distributed systems.
C	…guarantee failure transparency. False. Not all the standards are specified for distributed systems. Without using replicated components some failure scenarios cannot be managed. So, even those standards that belong to the distributed computing field may not be able to deal with all types of failure.
D	…improve throughput. False. Many standards provide an efficient solution to a given problem, but there might be other non-standardised solutions with better throughput. So, standards may not provide the best solution considering throughput.
E	All the above.
F	None of the above.

26. From a programmer's point of view, when a standard is followed…

A	…the programmes are easy to write, since there is a lower complexity in the handled elements.
B	…the final result is more reliable, since the standard introduces clearly defined ways of doing things.
C	…the obtained code has an easy maintenance since, although standards also change, those changes usually guarantee backwards compatibility.
D	…the programmes are easy to write, since standards are based on well-defined and high-level concepts.
E	All the above. True. All these aspects were considered when standards were included in Unit 3. Standardisation and interoperability are two main targets of middleware layers. This justifies their description in Unit 3.
F	None of the above.

27. Two approaches of remote method invocation in the web services area are…

A	SOAP and REST. True. These have been the approaches presented in Unit 3. The student guide describes the latter.
B	ZeroMQ and nanomsg. False. They are examples of message-oriented middleware to be used in asynchronous communication.
C	RPC and RMI. False. They are remote invocation mechanisms for process- or object-oriented systems. They haven't been designed for web services.
D	Client-server and peer-to-peer. False. They are the general interaction patterns (or approaches) in the distributed computing domain.
E	All the above.
F	None of the above.

28. The REST architectural style…

A	Uses HTTP as its "transport".
B	Uses only four basic "methods": GET, PUT, POST and DELETE.
C	Uses its GET method for read-only actions.
D	Takes the client-server architectural style as its basis, and promotes the use of stateless servers (in order to easily overcome failures).
E	All the above. Those are the main characteristics of the REST architectural style. They were carefully described in the student guide of Unit 3.
F	None of the above.

29. Some examples of "other middleware" are…

A	gedit. False. It is a regular text editor for X-Window systems. As such, it is only an example of user-level application. It doesn't assume that it is being executed in a distributed system.
B	OAuth. True. It is an example of security-related middleware.
C	Linux. False. It is an example of operating system.
D	MS-DOS. False. It is an example of operating system.
E	All the above.
F	None of the above.

30. Naming middleware…

A	…ensures failure transparency. False. The naming service doesn't participate in a direct way in any failure management subsystem. If we assume that it collaborates in any such subsystem, it won't be the main component of that subsystem. Because of this, it doesn't ensure failure transparency by itself.
B	…provides location transparency. True. It is a key component for ensuring location transparency. With its help, clients may refer to external agents (i.e., servers) using a name. Such names are translated into addresses by the naming middleware. Thus, the actual address isn't known by those clients. This provides location transparency. DNS is an example of service of this kind. The CORBA Name Service and the Java RMI registry are other examples. They are needed for obtaining the object references that are later maintained in the proxies.
C	…implements stateless servers. False. If other servers are stateful or stateless doesn't depend on using any naming middleware.
D	…improves system scalability. False. Name resolution introduces an additional step (that usually requires communication with a remote entity) in order to reach the object, service or process being referred by a name. So, although naming services are convenient for providing location transparency, their results should be cached. Otherwise, they would increase coupling, penalizing system performance and compromising system scalability.
E	All the above.
F	None of the above.

31. **A *Service Level Agreement* (SLA) is…**

A	…an agreement between service providers and service customers.
B	…a specification of service characteristics (e.g., functionality, throughput, response time, availability…) and their levels to be guaranteed.
C	…one of the aspects to be considered for deciding the number of instances of each service component at deployment time.
D	…something to be considered in PaaS systems for filling the deployment and scaling plans for a given service.
E	All the above. All these previous sentences are true. Such information was provided in different parts of Unit 4. The SLA is an agreement settled between service providers and service customers. In an ideal scenario, that agreement should specify the expected quality of service to be guaranteed by the provider and the maximal workload to be introduced by the customers. Such information is used at deployment time to deal with the aspects described in items C and D.
F	None of the above.

32. **In Unit 4, a service is…**

A	…a distributed application that has been deployed and is active. True. This was the informal definition given in the introduction of Unit 4.
B	…a set of independent scripts with a deployment plan. False. In the regular case a service consists of multiple components that collaborate in order to reach a common goal. They cannot be independent, but highly related. They don't need to be scripts.
C	…a future distributed application that is still in its analysis or design stages. False. A service is a distributed application that is already running. It has successfully passed the analysis and design stages of its life cycle.
D	…a specific Node.js programme that is executed by a single user. False. In that case such programme doesn't require a distributed setting. In the scope of Unit 4, the "service" concept refers to distributed software services.
E	All the above.
F	None of the above.

33. These are some tasks to be considered when a distributed application is being deployed…

A	To decide how many instances of each component should be run, and where.
B	To decide which dependent services should be used by the distributed application being deployed.
C	To decide the order in which each service component should be started.
D	To contact the OS or container in each host to start its components.
E	All the above. True. All those tasks are examples of steps being needed for deploying a distributed application.
F	None of the above.

34. Service life cycle management is strongly related to deployment. Some of its tasks are…

A	Component upgrades.
B	Configuration changes.
C	Component failure detection and recovery.
D	Scale-out or scale-in decisions, depending on the current workload.
E	All the above. True. Again, all those tasks are included in the service life cycle management that has been described in Unit 4.
F	None of the above.

35. Some problems that arise when a regular application is deployed in a desktop computer are…

A	…software dependency resolution; i.e., to find the appropriate libraries such application depends on.
B	…to give appropriate values to the environment variables being used by such application, if any.
C	…to appropriately configure the application (e.g., via registry in Windows, configuration files in Linux, `/Library` files in Mac OS, etc.)
D	…to find out if the application requirements are met by the current state of the target computer and operating system.
E	All the above. True. All those are valid examples of aspects to be considered when an application is deployed onto a desktop computer. Part C is regularly handled by the programme installer and it is transparent for the user.
F	None of the above.

36. Some of the elements in a deployment descriptor are…

A	Naming middleware. False. It doesn't make sense to include a complete middleware (regularly, a large piece of software providing general services to many applications) in a deployment descriptor.
B	Client command (e.g., docker). False. A deployment descriptor provides information for driving the deployment tasks. The "client command" concept was explained in the scope of Seminar 6, when containers were described. It is not a part of a deployment descriptor.
C	Filled deployment plan. True. A filled deployment plan, some filled component configuration templates and some component dependency descriptions are the main parts of a deployment descriptor.
D	Dockerfile. False. The "Dockerfile" is a specialised configuration file to be managed in a Docker system. Such file contains the sequence of instructions to be executed for building a docker image. It is not any part of a general deployment descriptor.
E	All the above.
F	None of the above.

37. A component needs the following elements in order to be deployed…

A	Its programme (or BLOB).
B	A filled configuration template.
C	A description of all its dependences.
D	A specification of its endpoint.
E	All the above. True. All those elements are needed for deploying a component. B, C and D are parts of its deployment descriptor. Without A, nothing can be made.
F	None of the above.

38. Dependency injection…

A	…decouples the component code from any concrete implementation of dependences, and is supported by container environments. True. This is the definition of dependency injection given in Unit 4.
B	…requires the use of environment variables to resolve dependences. False. When dependency injection is used, dependencies are resolved plugging the appropriate software modules into a component. That should be the main mechanism for binding a component to another. Environment variables might not be required in that case.
C	…requires the use of configuration files for resolving dependences. False. When dependency injection is used, dependencies are resolved plugging the appropriate software modules into a component. That should be the main mechanism for binding a component to another. Configuration files might not be required in that case.
D	…solves all dependences statically at implementation time. False. Dependency injection is a dynamic mechanism that allows a late binding between components (i.e., at deployment time, instead of implementation time).
E	All the above.
F	None of the above.

39. In the IaaS service model…

A	…deployment is completely automatized by the cloud provider. False. This only happens in the PaaS service model, and it isn't yet a mature set of technologies in such service model.
B	…several initial deployment decisions aren't automatized: amount of component instances, type of VM required by each component… True. Those decisions should be taken by the customer in an IaaS service model.
C	…life-cycle related deployment decisions are automatized; e.g., which workload levels throw scaling out/in actions, how to upgrade component SW… False. This only happens in the PaaS service model, and it isn't yet a mature set of technologies in such service model.
D	…no deployment support is given by the provider. False. A minimal support is already provided; e.g., image deployment onto virtual machines.
E	All the above.
F	None of the above.

40. In Windows Azure, some aspects of its deployment support are…

A	There is a basic service upgrading plan, although it doesn't support stateful services.
B	Components are known as "roles".
C	There is a basic fault domain management that enhances service availability.
D	There is no deployment sequencing plan.
E	All the above. True. All those aspects were included in the description of Windows Azure given in the last section of Unit 4.
F	None of the above.