# ENGINEERING SOFTWARE PRODUCTS

## QUIZ BOOK

# Introduction

The quizzes in this book have been designed to support the teaching of software engineering courses based on the book. There is a quiz for each chapter and the answers to the questions in the quiz can be found in the book.

Software engineering practice does not lend itself to multiple-choice tests that can be automatically marked. Rather, each quiz question has a short answer (usually a few words or a single sentence).

Quizzes have been designed to allow readers to test their recall of what has been covered in a chapter. The answers to all of the questions can be found in the text and I recommend that students complete the quizzes after the material in a chapter has been covered. For native English speakers, it should not take more than 20 minutes to complete each quiz. Slightly longer should be allowed if students are not completely fluent in English.

Quizzes may either be closed book or open book but I recommend if open book then the time made available should be reduced.

The quiz book has been deliberately made available in MS Word format rather than PDF so that instructors may reorganize and present the material in a form that is suitable for their classes.

**All quizzes and answers are ©Ian Sommerville 2019. You may modify and reformat the quiz questions as you require but you should not disseminate them except for the purposes of teaching and instruction.**

**Please do not make the answers to the quizzes available on a public server.**

# Quiz Ch 1: Software Products

1. How are the requirements for a software system established in a project-based software engineering process?
2. Why is it important that software products are brought to market quickly?
3. What is a software product line?
4. List three types of software development that have much in common with software product development.
5. What is a product vision?
6. What fundamental questions should be answered in a product vision?
7. List the principal information sources that may be used in developing a product vision.
8. What is software product management?
9. What factors must be considered by software product managers when they are doing their job?
10. What is a product roadmap?
11. What is a product backlog?
12. Why is it important to develop a prototype as the first stage of product development?

## Ch 1: Answers to quiz questions

1. The software requirements are established by the customer for the software based on the problems they wish to address and their current processes (and existing systems).
2. Customers are reluctant to change after they have made product decisions. Therefore, if a competitor is first-to-market, it is much more difficult to establish a customer base for the product being developed.
3. A software product line is a software system (usually a large system) that is adapted to the needs of specific customers by making changes to part of the system. Consequently, each customer has a different version of the product.
4. Student projects, internal tool development, research software.
5. A succinct statement that defines the essence of the product that is being developed. It should explain how the product differs from competing products.
6. What is the product that is proposed? Who are the target customers and users? Why should customers buy this product?
7. Domain experience, product experience, Customer experience, prototyping and ‘playing around’.
8. Software product management is a business activity focusing on the software products that are developed and sold by the business. Product manag-ers (PMs) take overall responsibility for the product and are involved in planning, development, and marketing.
9. Business needs, technology constraints, customer experience.
10. A plan for the development, release and marketing of a software product.
11. A to-do list that sets out what has to be done to complete the product development.
12. To check what you want to do is feasible and to demonstrate the software to prospective customers and company funders.

# Quiz Ch 2: Agile Software Engineering

1. What are the three main objectives of agile software engineering?
2. What is incremental development and delivery?
3. List five agile development principles.
4. What are the most widely adopted Extreme Programming practices?
5. How are XP teams managed?
6. What is Scrum?
7. What is the principal responsibility of the Product Owner in a Scrum team.
8. What is a ‘potentially shippable product increment’?
9. List five benefits of using Scrum.
10. What are the principal product backlog activities in Scrum?
11. What factors should be taken into account when deciding how many story points are required to complete a backlog item.
12. What is meant by a Scrum development team’s velocity?
13. Briefly explain the three main benefits of using timeboxed sprints.
14. What three things should a Scrum team do when planning a sprint?
15. What is a Scrum?
16. What technical activities from XP should always be used in a Scrum development process?
17. List the items that might be part of a code completeness checklist.
18. What is a self-organizing team?
19. Why does informal verbal communication not always work as a means of team coordination?
20. What are the three principal project management activities? For each of these, list two associated activities that are management responsibilities.

## Ch 2: Answers to quiz questions

1. Deliver functionality quickly, respond quickly to changing product specifications, minimize development overheads.
2. The development and delivery of a software product in discrete stages where, in each stage, some of the product features are implemented and tested.
3. Involve the customer, embrace change, develop and deliver incrementally, maintain simplicity, focus on people not process.
4. Incremental planning/user stories, Small releases, test-driven development, continuous integration, refactoring.
5. Management is a collective team activity, with no designated project manager.
6. Scrum is a framework for agile project organization.
7. The Product Owner is responsible for ensuring that the development team are always focused on the product that they are building rather than diverted into less relevant technically interesting work.
8. A code increment that is of product-quality i.e. the code has been completely tested, reviewed and documented.
9. Unstable requirements don’t hold up progress, stakeholders can relate to product increments, customers see on-time delivery of increments, team communication is improved, trust is established between developers and customers.
10. Refinement, estimation, creation, prioritization.
11. The size of the task, its complexity. the technology required and the unknown characteristics of the work.
12. A team’s velocity is the sum of the size elements that have been completed during a fixed-length sprint.
13. Demonstrable progress where there is a tangible output at the end of each sprint, work planning where the team understand how much work can be completed, problem discovery so that rework required is limited to the duration of a sprint.
14. Establish an agreed sprint goal, decide on PBIs that should be implemented, create a sprint backlog.
15. A scrum is a daily meeting where the team discuss what has been achieved and coordinate their work for that day.
16. Test automation and continuous integration
17. Reviewed, unit tested, integrated, integration tested, accepted.
18. A development team that makes their own decisions about work allocation, coordination, schedule and external interactions.
19. Team members may not work full-time, team members may work remotely, team members may work on several projects at once and so cannot be available for daily scrums.
20. Reporting to company management (risk reporting, progress reporting, budget drawdown), administration (managing compliance, procurement of equipment and materials, approving expenditure), people management (managing absence, work reviews, hiring)

# Quiz Ch 3: Features, Scenarios and Stories

1. What factors drive the design of most software products?
2. What is a ‘feature’ of a software product?
3. Explain how personas, scenarios and user stories can contribute to feature design.
4. What is a persona?
5. What information should be included in a persona description?
6. What is a scenario?
7. What are the most important elements that may be included in a scenario description?
8. What is the main problem with using a structured approach to scenario description?
9. What are the problems of writing goal-based scenarios?
10. How can users get involved in the development of scenarios?
11. What is a user story?
12. Show a typical form of a user story and explain its constituent parts.
13. Why should you use user stories as well as scenarios when deriving a feature list for a software product?
14. Explain why you should normally avoid writing ‘negative stories’.
15. What are the most important characteristics of software features?
16. List six important factors in feature set design.
17. What is ‘feature creep’?
18. Explain how you can identify features from a scenario description.
19. Explain how features can be described and defined using user stories.
20. Explain why user research on its own is not necessarily sufficient when designing software products.

## Ch 3: Answers to quiz questions

1. Business and consumer needs that are not met by current products, Dissatsfaction with existing products, technology changes that make new types of product possible.
2. A feature is a fragment of functionality that is offered by a software product.
3. Personas inspire scenarios; scenarios are developed into stories and inspire feature; stories define features.
4. A persona is an imagined user where you create a character portait of a type of user that might use your product.
5. Information about the person themselves, their job, their educational background and experience and why they might be interested in using the product.
6. A narrative that describes how a user or group of users might use a software system to do something that they want to do.
7. Scenario name, overall objective, what’s involved in reaching the objective, personas of actors involved, problem that can’t be addressed by existing system, possible ways that problem can be tackled and their may be a requirement to include a very specific feature in a system.
8. Users who have to understand scenarios find structured descriptions intimidating and hard-to-understand.
9. It is often easier for users to articulate how the do some task using an existing system, rather than provide a more abstract description of their goals. Sometimes, it makes sense to include implementation details as the scenario involves using well-known universal functionality such as cut and paste.
10. Users are best involved as critics of a scenario rather than as scenario writers. Based on an initial scenario, users can point out what is wrong, suggest how it can be extended and ask questions about aspects they don’t understand.
11. A user story is a fine-grained narrative that sets out, in a structured way, a single thing that a user wants from a software system.
12. As a <role> I <want|need> to <do something> so that <reason>
13. User stories add detail to a scenario and can serve as a system feature description. They are sufficiently detailed that they can be used for planning the system implementation.
14. You should avoid negative stories because it is impossible to write tests that conclusively demonstrate a negative.
15. Independence, coherence and relevance.
16. Simplicity, functionality, familiarity, novelty, automation, control.
17. The number of features in a product ‘creeps up’ as new potential uses of the product are envisaged.
18. Highlight key phrases in the interactions description that describe specific software functionality that is used.
19. A feature description should include an overall description of the feature, a set of user stories where each story describes different aspects of the feature, constraints on the feature implementation and a comments section for any other information that may be useful when implementing the feature.
20. User research shows how people do things at the moment and tends to lock in existing ways of working. It does not take into account the possibility of product innovations changing the ways that things are done.

# Quiz Ch 4:

1. Why is software architecture important?
2. What is the IEEE definition of software architecture?
3. What are the differences between a centralized and a distributed security architecture .
4. List the most important architectural issues for product development
5. What architectural choice is likely to increase system maintainability
6. Explain what is meant by system availability
7. What are the three fundamental questions you should consider during the architectural design process?
8. List 4 types of software component relationships.
9. Suggest three ways of controlling architectural complexity.
10. Why is is sometimes impossible to localise component interactions within a single layer?
11. What are cross-cutting architectural concerns?
12. What types of integration are possible when multiple services are included in a software product?
13. Why do you need to consider the technologies used in a layered architecture when decomposing a system into components?
14. Briefly describe the notion of a client-server architecture.
15. What is the function of the fundamental components of the MVC pattern?
16. What is JSON and how does it differ from XML?
17. What is a service-oriented architecture?
18. List 5 technology choices that you have to make when designing a software architecture
19. What factors have to be considered when implementing a mobile product?
20. What are the advantages and disadvantages of using open-source software.

## Ch 4: Answers to quiz questions

1. Architecture is important because it affects the performance, usability, security, reliability and maintainability of a software system.
   1. Architecture is the fundamental organization of a software system embodied in its components, their relationships teach other, and to the environment and the principles guiding its design and evolution.
   2. In a centralized security architecture, all assets are stored in one place with layers of security protection around them. In a distributed architecture, assets are stored in different places so that a single successful attack does not gain access to all assets.
   3. Non-functional product characteristics, product lifetime, software reuse, number of users, software compatibility.
   4. Constructing a system from a number of fine-grain interacting components where each component has a single responsibility.
   5. Availability is expressed as the percentage of time when a system is up and running and available to deliver its services.
   6. How should a system be organized as a set of architectural components? How should components be distributed and communicate? What technologies should be used to build the system.
   7. Part-of, uses, is-located-with, shares-data-with.
   8. Components should focus on implementing a single concern, functionality should not be duplicated at different places in a system, component interfaces should be stable.
   9. For efficiency reasons, it may be necessary to divide functionality so that is implemented in separate layers (otherwise functionality would be duplicated). Some concerns are cross-cutting and cannot to localized within a layer.
   10. These are concerns, such as security and performance, that cannot be localized to a single layer in the system architecture.
   11. Full integration where services are aware of and can communicate with other services through their API, partial integration where services share components and databases but do not interact directly and independent where there are no shared services or databases.
   12. The choice of technologies used in a layer affects the components in the layer above.
   13. A user interacts using a client computer or mobile device, which hosts the user interface for the system; system functionality is implemented on a remote server. Communication now is usually based on the http protocol.
   14. Model , which manages the data that is to be displayed; View, which manages the representation or representations of the data as presented to the user; Controller, which manages changes made to the model and communicates with the View to ensure the representations are updated.
   15. JSON is a language for describing data that is communicated between components in a system; it differs from XML in that it is based on a structured organisation without the need for explicit tags for every data item.
   16. An architecture where the fundamental software components are stateless, independent software services.
   17. Database, platform, server, open source, development tools
   18. Intermittent connectivity, processor power, power management, on-screen keyboard.
   19. Advantages: lower cost, tested software and faster time to market; disadvantages are lack of control over software evolution, lack of competitive edge and possibly restrictive license conditions.

# Quiz Ch 5: Cloud-based software

1. What do you understand by the terms scaleability and resilience?
2. List 4 benefits of using the cloud for software development.
3. What makes it possible for software running on a virtual server to ‘run anywhere’?
4. What are containers in cloud computing?
5. List the elements of the Docker container system.
6. What are the four most important benefits of using containers?
7. What the advantage of using PaaS if you are developing cloud-based software?
8. What is ‘function as a service’?
9. List 2 advantages and 2 disadvantages of SaaS for customers.
10. What are the design issues that you must consider when implementing SaaS?
11. What is a multi-tenant database?
12. List five possible customizations for SaaS.
13. What are the problems of providing database extensibility by adding additional fields?
14. What are multi-instance systems?
15. List the advantages and disadvantages of multi-instance databases.
16. What are the key architectural decisions for cloud software engineering?
17. What are the key questions to ask when choosing a database organization?
18. Briefly explain how a cloud-based system can be resilient.
19. List three technical and three business issues that should influence cloud platform choice.
20. What is a service-level agreement (SLA)?

## Ch 5: Answers to quiz questions

1. Scaleability - the ability to cope with increasing demand without loss of service or performance; resilience - the ability to continue to deliver service in spite of software or hardware failures.
2. No capital costs, no delays in setting up servers, choice of server types, ability to do distributed development.
3. All of the software that is required is loaded onto the virtual server so the unit of portability is the server rather than the individual program.
4. A lightweight operating system virtualisation technology that can contain all software needed to run a program.
5. Docker daemon, Docker client, Dockerfiles, Image, Docker hub, Containers.
6. Fast startup time, mechanism for software portability across clouds, efficient mechanism for running services, support DevOps.
7. You can use functionality supplied by the cloud provider that makes it easy to scale your software (up or down) and make it resilient.
8. A single function is implemented as a service with no need for an explicit server; the cloud provider starts up the function when required and shuts it down when it is no longer needed.
9. Advantages:No upfront costs for software or servers, Immediate software updates, reduced management costs.

Disadvantages: Service lock-in, possibly slow data transfer, privacy regulation issues.

1. Local/remote processing, authentication, information leakage, multi tenant or multi-instance database management
2. A database where information for all customers is held in the same database and identified using some customer identifier.
3. Authentication, branding, business rules, data schemas, access control
4. How many extra fields should be included? How to provide typing when different customers need to store different types of data in each additional field.
5. SaaS systems where there is a separate database for each customer.
6. Advantages; more flexible, more secure, more easily scaled, easier to provide resilience; Disadvantages: higher costs, slower and more expensive update management.
7. Database: multi-tenant or multi-instance; Scaleability: what are scaleability and resilience requirements; Software structure: monolithic or microservices
8. Who are target customers, are ACID transactions needed, how big will the database be, will there be a need to transfer info from one database to another, can the DB be organized as a set of smaller databases for microservices.
9. Replicate software in containers across different physical servers; provide mechanisms such as database mirroring, to switch to a replica in the event that a monitoring system detects failure.
10. Technical: Resilience, privacy and data protection, expected load; Business: Cost, developer experience, service-level agreements
11. An agreement between a cloud user and cloud provider that sets out the expected level of service from the cloud provider.

# Quiz Ch 6: Microservices

1. What is a microservice?
2. List five characteristics of microservices.
3. What is coupling and why is it important in a microservices architecture?
4. What is the single responsibility principle?
5. Explain what is meant by the ‘rule of twos’.
6. What support code is needed in a microservice?
7. What fundamental problems are addressed by the microservices architecture style?
8. List 3 key design questions that should be addressed when designing a microservices architecture.
9. What are four general design guidelines that support the decomposition of a system into microservices.
10. What is the difference between synchronous and asynchronous microservices interaction?
11. Explain what is meant by indirect service communication.
12. Explain what is meant by replica inconsistency.
13. What is a compensating transaction and when are compensating transactions used.
14. What is eventual consistency?
15. Explain the difference between orchestration and choreography.
16. List three possible failure types in a microservices architecture.
17. How does a circuit breaker work?
18. In the RESTful style, what is a resource?
19. What are the four fundamental RESTful service operations and how do they map to HTTP verbs?
20. Why should you use versioned services in a microservices architecture?

## Ch 6: Answers to Quiz questions

1. A small-scale, stateless software component that is completely independent and which implements a single business function.
2. Self-contained, lightweight, Implementation-independent, independently deployable, business-oriented.
3. Coupling is a measure of the number of relationships between components. To be independent, components should have a low coupling.
4. Each element in a system should do one thing and should do it well.
5. It should be possible for a microservice to be developed in two weeks; a microservice development team should be sufficiently small that it can be fed using two large pizzas.
6. Message management code, failure management code, UI implementation code, data consistency management code.
7. Only those parts of a system that have been changed need to be re-tested and redeployed; it is not necessary to scale the whole system when the demand on parts of the system increases.
8. Any three from: what are the microservices that make up the system; how should microservices communicate; how should service failure be detected and managed; how should microservices be coordinated; how should data be distributed and shared.
9. Balance fine-grain functionality and system performance; follow the common closure principle; associate services with business capabilities; design services so that they only access the data that they need.
10. Synchronous interaction: a microservice makes a request to another service and waits for its reply; asynchronous interaction: when a service request is made, the requesting service continues processing and does not wait for a reply.
11. Indirect services communication means that a requesting service does not request the service using its address (URI) but rather requests the service by name. A service broker is responsible for passing the service request to the named service and returning a reply to the requesting service.
12. Replica inconsistency means that different instances of the same service have inconsistent databases.
13. A compensating transaction is a transaction that reverses a previous operation. They are used when one of the services in an interaction fails after other services have changed their database to restore consistency.
14. Eventual consistency means that it is guaranteed that all of the databases in replica services will eventually become consistent.
15. In an orchestrated system, there is a single service controller that is responsible for managing service interactions; in a choreographed system, there is no central controller and each service emits events to indicate it is finished processing. Services that need to coordinate, watch for these events and react when they are observed.
16. Internal service failure, external service failure, service performance failure.
17. Service requests are routed through the circuit breaker that monitors requested services to check they are available. If a service is unavailable, the service request is rejected immediately without the need for the requesting services to wait to see if the request has been processed.
18. A resource is a uniquely addressed information item that this accessed through its URI.
19. Create (HTTP POST), Read (HTTP GET), Update (HTTP PUT), Delete (HTTP DELETE).
20. Versioned services should be used so that, in the event of a service failure after an update, requests can be routed to a previous version of the service; they also allow services that rely on features of an older service version to continue operation until they are updated.

# Quiz Ch 7: Security and Privacy

1. List three types of security threat.
2. List four types of management procedure that are needed to maintain overall system security.
3. Suggest three features that may be included in cloud-based systems to help users with operational security.
4. What is an injection attack?
5. How does a cross-site scripting attack work?
6. What is session hijacking?
7. What is a distributed denial of service attack?
8. List three ways of authenticating a user of a software product.
9. What are the major weaknesses of password-based authentication?
10. Explain what is meant by ‘federated identity’.
11. What is an ‘access control list’?
12. What is the difference between symmetric and asymmetric encryption?
13. Why do we continue to use symmetric encryption?
14. List the five main elements of a digital certificate.
15. What are the four different levels in a system where data may be encrypted?
16. What are the major drawbacks of application-level encryption?
17. Briefly explain what is meant by ‘key management’.
18. What is ‘privacy’?
19. What areas may be covered by data protection laws?
20. List 5 data protection principles that underlie the GDPR.

## Quiz Ch 7: Answers to quiz questions

1. Threats to the availability of a system, threats to the integrity of a system or its data, threats to the confidentiality of the data managed by a system.
2. Authentication and authorisation management, system infrastructure management, attack monitoring, backup policies and management.
3. Auto-logout, user command logging, multi-factor authentication.
4. A type of attack where a malicious user uses a valid input field to input malicious code or database commands, which are aimed at causing some damage to the system.
5. An attacker introduces malicious code into a legitimate website using some security weakness. When a valid request is made to that website, the malicious code is executed and information, such as user keystrokes, are sent from the user’s browser to the attacker.
6. Session hijacking is a type of attack where authentication information set up in a user session (session cookie) is stolen by an attacker who uses this to impersonate a legitimate user.
7. A distributed denial of service attacker involves a network of remote computers flooding a legitimate site with requests so that it is overloaded and cannot deliver normal service.
8. Something the user knows such as a password, something the user owns, such as a mobile phone, some attribute of a user such as a fingerprint.
9. Insecure passwords, phishing attacks, password reuse, forgotten passwords.
10. Federated identity is an approach to authentication where an authenticating site relies on an external service, such as Google, to authenticate a user.
11. An access control list is a list of user permissions that sets out the access to system resources that is allowed for each user.
12. In symmetric encryption, the same key is used to encrypt and decrypt confidential information; in asymmetric encryption, a different key is used for encryption and decryption.
13. Symmetric encryption is widely used because it is much faster than asymmetric encryption.
14. Subject information about the certificate holder, certificate authority information, certificate information, digital signature of the certificate, public key information for the certificate holder.
15. Media level (eg disk encryption), file level, database level, application level.
16. Most software engineers are not encryption experts and so can make mistakes in encryption implementation, encryption and decryption slows down application performance, encryption keys must be carefully managed in a KMS.
17. Key management means generating and securely storing encryption keys and managing these keys over time. They must be linked to the right version of the encrypted information.
18. Privacy is a social concept that relates to the collection, dissemination and use of personal information held by a third-party.
19. Responsibities of data controllers such as data storage, data use, security and subject access. Rights of data subjects including access rights, error correction, data deletion and usage consent.
20. Any five from: users must be aware of what data collected and have control over its use, the purpose for which data is collected and stored must be explained, user consent for data storage must be granted, data must only be stored for as long as it is required, data must be stored securely, users must be able to find out what information is stored and be allowed to correct errors, data must not be stored in countries with weaker data protection laws.

# Quiz Ch 8: Reliable programming

1. What are the major quality attribute groups and what attributes are in each group?
2. List three low-cost techniques for software reliability improvement.
3. Explain what is meant by ‘fault avoidance’.
4. Give three underlying causes of program errors.
5. What is program complexity?
6. Why does increased complexity lead to program errors?
7. What are the three main types of program complexity?
8. Give 2 examples of complexity reduction guidelines for each of the types of program complexity that you have identified in question 7.
9. Why is it important to avoid deep inheritance hierarchies?
10. What is a ‘design pattern’?
11. List three possible classification groups for design patterns.
12. What is ‘program refactoring’?
13. Give four examples of ‘code smells’.
14. What is meant by ‘input validation’?
15. List four ways of implementing input validation.
16. What is a regular expression and how is it useful in input validation.
17. Why is number checking important?
18. What are the three most important categories of software failure?
19. What is meant by a program ‘exception’ and ‘exception handler’?
20. Explain two mechanisms that can be used to help users recover work after a system failure.

## Quiz Ch 8: Answers to quiz questions

1. Reliability attributes (reliability, availability, security, resilience), User experience attributes (responsiveness, usability), maintainability.
2. Fault avoidance, input validations, failure management.
3. Fault avoidance is an approach to programming whose aim is to avoid faults being introduced into a program and, consequently, to reduce program failures.
4. Programmers make mistakes because (1) they don’t completely understand the problem they are trying to solve (2) they use unsuitable technology or don’t understand the technologies used (3) they make simple slips or do not understand the implications of component interactions.
5. The complexity of a program depends on the number of relationships between elements in the program and the type and nature of these relationships.
6. Increased complexity leads to errors because our brains can only process a limited number of information quanta in short-term memory. The higher the complexity, the more information must be transferred between short and long-term memory and errors are likely to arise in this information transfer.
7. Structural complexity, data complexity, conditional complexity.
8. Structural complexity (functions should do one thing and one thing only, functions should never have side-effects), data complexity (define interfaces for all abstractions, define abstract data types), conditional complexity (avoid deeply nested conditional statements, avoid complex conditional expressions). Other guidelines from Table 8.1 also possible.
9. You should avoid deep inheritance hierarchies because you have to examine all classes higher in the hierarchy when making changes to an object class at a low level. This involves more information processing and so is potentially error-prone.
10. A general reusable solution to a commonly-occurring problem within a given context in software design.
11. Creational patterns, structural patterns, behavioural patterns.
12. Changing a program to reduce its complexity without changing the external behaviour of the program.
13. Any four from: large classes, long methods/functions, duplicated code, meaningless names, unused code.
14. Input validation means checking every input to a program to ensure that it is in the correct format and within the range defined by input rules.
15. Built-in validation functions, type coercion functions, explicit comparisons, regular expressions.
16. A regular expression is a definition of a text pattern. To use in input validation, you define a regular expression based on the syntax rules for expected inputs and check that inputs match that pattern.
17. Number checking is important because numbers that are too large or too small may lead to unpredictable program behaviour and because invalid values for a number ‘pollute’ a database and affect other programs/functions using that information.
18. Data failures, program exceptions, timing failures.
19. A program exception is an unexpected or abnormal event that occurs during program execution; an exception handler is a program unit that includes code to process such unexpected events.
20. Activity logging and auto-saving user data during an interaction sessions.

# Quiz Ch 9: Testing

1. List two causes of program bugs.
2. Define what is meant by ‘functional testing’.
3. Apart from functional testing, list three other types of program testing.
4. What are the two phases of user testing?
5. What general principle underlies unit testing?
6. What is an equivalence partition? Give an example.
7. List five unit testing guidelines.
8. What are the two types of test involved in feature testing?
9. What four things is system testing concerned with?
10. What is an end-to-end pathway?
11. What do you understand by an ‘executable test’?
12. Briefly outline a commonly-used structure for an executable test.
13. What is regression testing?
14. How do interaction recording tools support system testing?
15. What is test-driven development?
16. What are the benefits of test-driven development?
17. List the three major challenges of security testing.
18. Give five examples of security risks.
19. What are three fundamental problems of testing?
20. Briefly outline the code review process.

## Quiz Ch 9: Answers to quiz questions

1. Programming errors and understanding errors
2. Testing the functionality of a program to find bugs and to demonstrate that the software works as expected.
3. User testing, performance and load testing, security testing.
4. Alpha testing, where the aim of testing is to answer the question ‘do users want the planned system features?’ and beta testing where users test the usability of a product.
5. If a program behaves as expected for a set of inputs with some shared characteristics, it will behave in the same way for a larger set whose members share these characteristics.
6. A set of inputs that have some common characteristic, such as positive integers less than 1000.
7. Any five from: test edge cases, force errors, fill buffers, repeat yourself, overflow and underflow, don’t forget null and zero, keep count, one is different.
8. Interaction tests and usefulness tests.
9. Testing to discover feature interactions, testing to discover if features do what users want, testing the system in different operational environments, testing system responsiveness, security, etc.
10. A sequence of user events that represent a user interaction from start to finish. Typically, these will use several system features.
11. An executable test is a program component that can be executed and which can automatically determine if the test has been successful.
12. Arrange - set up the system to run the test; Action - call the unit being tested; Assert - check the test result against the expected result.
13. Regression testing is the process of running existing tests when a change has been made to a program to check that the change has not adversely affected previously tested functionality.
14. They record mouse movements and clicks, menu selections and keystrokes. These can then be replayed to repeat the test. They also allow test scripts to be written and executed.
15. TDD is the process of writing unit tests before the functionality of the program unit is defined. The unit is then modified so that it passes the previously written test. This means that a unit should always pass all previously written tests.
16. Tests are clearly linked to code sections, tests act as a specification for the code, debugging is simplified because it is linked to the most recently written code, TDD may lead to simpler code.
17. It involves testing for something that the system should not do, vulnerabilities often lurk in rarely used code so are not revealed in functional tests, vulnerabilities may come from external software such as browsers, etc.
18. Any five from: Attacker gains access using authorised credentials, authorised user accesses forbidden resources, authentication system does not identify attacker, attacker gains database access using SQL poisoning, HTTTP sessions are improperly managed, session cookies revealed to attacker, confidential data is unencrypted, encryption keys are insecurely stored.
19. Testing depends on the tester’s understanding of what the code should do and this may be incorrect, tests are sometimes difficult to design so test coverage is incomplete, testing does not tell you anything about program attributes such as readability or structure.
20. Setup review and distribute code for review (programmer), check code and write review report (reviewer), discuss review suggestions (programmer and reviewer), made code changes (programmer)

# Quiz Ch 10: DevOps and code management

1. What do you understand by the term ‘DevOps’?
2. What three factors led to the development and adoption of DevOps?
3. What are the three principles that are the basis of Devops?
4. List four important benefits of Devops.
5. What is code management?
6. What general areas are supported by code management systems?
7. List five features of code management systems.
8. What are three advantages of distributed code management systems?
9. What are ‘branching and merging’?
10. What are the four aspects of DevOps automation?
11. What is the main purpose of an issue management system?
12. What is meant by continuous integration?
13. How does incremental system building work?
14. What is continuous delivery and how does it differ from continuous deployment?
15. List four benefits of continuous deployment.
16. What is meant by ‘infrastructure as code’?
17. What are the key benefits of representing your infrastructure as code?
18. What are the four types of measurements that may be used in software development?
19. Why is it unreliable to link business success measurements with other software measurements?
20. List five metrics that may be used when assessing DevOps processes.

## Quiz Ch 10: Answers to quiz questions

1. DevOps (development+operations) integrates development, deployment and support, with a single team responsible for all of them.
2. Agile methods reduced software development time but deployment and delivery remained a bottleneck, Amazon experimented with integrating service development and support, software as a service became a realistic deployment model.
3. Everyone is responsible for everything, everything that can be automated should be automated, measure first, change later.
4. Faster deployment, reduced risk, faster repair of bugs and outages, more productive teams.
5. A set of software supported practices that are used to manage an evolving codebase, keeping track of changes made and ensuring the developers can work independently on the same code without interference.
6. Code transfer, version storage and retrieval, merging and branching, version information.
7. Version and release identification, change history recording, independent development, project support, storage management.
8. Resilience, speed, flexibility.
9. Branching occurs when a developer makes changes to a shared codebase and an independent branch is created to allow these changes to be made without interfering with other developers. Merging is the process of integrating the changes made in a branch into the shared codebase.
10. Continuous integration, continuous delivery, continuous deployment, infrastructure as code.
11. To keep track of issues and problems raised by developers and customers and the development teams responses to the issues that have been raised.
12. Continuous integration means that every time a developer makes a change to a code unit, it is immediately integrated into the shared codebase.
13. Incremental system building means that only those parts of a system that have been changed are recompiled and rebuilt. It relies on a dependency graph that shows code unit dependencies and time stamps that show when code units have been changed.
14. Continuous delivery is the processes of creating a deployable version of a system every time a change is integrated into the project codebase. It differs from continuous deployment in that deployment involves pushing that change to software customers; In continuous delivery, there is no requirement to release the changed software.
15. Reduced costs, faster problem solving, faster customer feedback, A/B testing.
16. An executable model of software infrastructure (databases, libraries, browsers, etc.) is maintained so that it is possible to recreate this specific infrastructure on demand.
17. Visibility - the model makes the infrastructure understandable; reproducibility - it is always possible to recreate the infrastructure that software relies on; reliability - human errors are reduced when updating the infrastructure; recovery - it is fast to rebuild a specific infrastructure after a problem has occurred.
18. Process measurements, service measurements, usage measurements, business success measurements.
19. Because business success is a complex issue that depends on many factors, such as management skills, that are nothing to do with software development. We do not know how to assess the contribution of software development changes in business success.
20. Any five from: mean time to recovery, percentage of failed deployments, deployment frequency, change volume, lead time from development to deployment, percentage increase in customer numbers, number of customer complaints, availability, performance.