**BCS Level 3 Certificate in Programming**

**QAN 603/1192/7**

**Version 1.0**

**March 2017**

By

Chatukuta, Tadiwanashe

Partington, Ben

Cronk, Sean

**Syllabus**

For each top-level area of the syllabus a percentage and K level is identified.

The percentage shown relates to the proportion of the exam given to that area.

The K level identifies the maximum knowledge level that may be tested for that area.

**1. The end-user context for software development activities (7.5%, K1)**

In this topic, the apprentice will learn the importance of understanding context before starting to create software. They will be aware of how to gain an understanding of end-users and the environment in which the software will be used (e.g. by a doctor in a hospital, or by a consumer through a web-site, or by an engineer in a manufacturing plant).

The successful apprentice should be able to:

**1.1 Outline the discovery phase of a project, determining the Why, Who, What, How and When.**

**1.2 Identify how user research can facilitate understanding of the end-user; who they are, what is the problem they are faced with and how they are currently dealing with the problem; including but not limited to:**

* Questionnaires
* User Interviews
* Contextual Enquiry
* Focus Groups
* Personas
* Customer Journey Mapping

**Questionnaires**

Questionnaires are used to determine what your development will do, how it will look, and how it will work.

<https://www.dogmasystems.com/5-questions-discuss-client-software-development-company/>

<http://www.rickquatro.com/forms/ClientQuestionnaire.pdf>

**User Interviews**

User interviews are a way of speaking to the customer about the software. This can be done with Questionnaires and Focus Groups.

<https://www.interaction-design.org/literature/topics/user-interviews>

<https://www.interaction-design.org/literature/article/how-to-conduct-user-interviews>

**Contextual Enquiry**

Collecting information regarding the topic, get knowledge on stakeholders. The data collected is contextual data, and the enquiry carried out is contextual enquiry.

<https://www.hcltech.com/blogs/essence-contextual-enquiries>

<https://chaione.com/blog/contextual-inquiries-what-how-why/>

Focus group

A focus group is a small group of individuals tasked by you to review your software in a critical manner, meaning they will critique in a honest manner and share their thoughts and opinions in a open honest matter.

<https://www.experienceux.co.uk/faqs/what-are-focus-groups/>

<https://www.linkedin.com/pulse/what-process-focus-group-software-development-lambros-photios>

<https://medium.com/stationfive/what-is-a-focus-group-in-software-development-and-why-do-you-need-one-c245b3600cb8>

**Personas**

Personas are fictional characters, which you create based upon your research in order to represent the different user types that might use your service, product, site, or brand in a similar way. Creating personas will help you to understand your users’ needs, experiences, behaviours and goals.

<https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them>

<https://www.usability.gov/how-to-and-tools/methods/personas.html>

**Customer Journey**

A customer journey map is a very simple idea: a diagram that illustrates the steps your customer(s) go through in engaging with your company, whether it be a product, an online experience, retail experience, or a service, or any combination.

<https://boagworld.com/usability/customer-journey-mapping/>

<https://hbr.org/2010/11/using-customer-journey-maps-to>

**1.3 Outline the ways in which individual, business and external constraints and dependencies such as those listed below can impact the delivery of a software project.**

**Compliance**

In general, compliance means conforming to a rule, such as a specification, policy, standard or law. Regulatory compliance describes the goal that organizations aspire to achieve in their efforts to ensure that they are aware of and take steps to comply with relevant laws, policies, and regulations.

<https://en.wikipedia.org/wiki/Regulatory_compliance>

<https://searchdatamanagement.techtarget.com/definition/compliance>

**Governance**

Software Engineering Governance is a component part of Corporate Governance - the set of structures, processes and policies by which an organisation is directed and controlled.

<https://dzone.com/articles/importance-governance-software>

<https://www.ibm.com/developerworks/rational/library/jun07/kroll/index.html>

**Ethics**

Almost any professional whose work impacts the public must abide by some sort of ethical code of conduct.

<https://www.computer.org/web/education/code-of-ethics>

**Legality**

When discussing and organizing software development teams, there are some principles, sometimes called laws, which teams need to be aware of.

<https://exceptionnotfound.net/fundamental-laws-of-software-development/>

<http://www.methodsandtools.com/archive/softwarelaws.php>

**Software Development Roles**

Project Sponsor

The project sponsor is the person or group that provides direction and resources, including financial resources for the software project

Subject Matter Experts (SME

A person who is an authority in a particular area or topic. Eg. accountants, finance controllers, salespeople, production managers

Project Owner

This is the person who represents the business or end-users and is responsible for working with the user group to determine what features will be in the product release.

Project Manager (PM)

This person is responsible for knowing the “Who, What, Where, When, and Why” of software project.

Technical Lead

This person is the development team leader. He translates the business requirements into a technical solution.

Software Testers

This person ensures that the software solution meets the business requirements and that it is free of bugs, errors and defects.

Release Engineer

This person deals with the accumulation and delivery of source codes into software or programs. They also oversee the development of the software

User Acceptance Testers (UAT)

This is the final step prior to a new software solution being released to production. It is often UAT that creates the bottleneck between the software solution being completed and released to the business.

**2. How code integrates into the wider project (10%, K2)**

In this topic, the apprentice will learn about the non-coding concepts integral to each stage of the software development lifecycle. They will also understand how software components are managed and controlled and how these are brought together into software solutions. This includes the knowledge of team-work and team structures.

The successful apprentice should be able to:

**2.1 Summarise the key business concepts and artefacts that must be considered during a software project. Including but not limited to:**

* Processes and Procedures (e.g. Business Process Management, Release Management)
* Documentation
* Training
* Support
* Service levels

**Processes and Procedures**

A software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management.

<https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc>

<https://www.itemis.com/en/agile/scrum/compact/fundamentals-of-project-management/procedure-models-in-software-development>

**Documentation**

In computer hardware and software product development, documentation is the information that describes the product to its users. It consists of the product technical manuals and online information

<https://www.altexsoft.com/blog/business/software-documentation-types-and-best-practices/>

<https://en.wikipedia.org/wiki/Software_documentation>

**Training**

Software development training can be one of two things. One could be the training of a programmer to learn which language to use when making the software. And it could also be, the training for the user on how to use the software being made.

<https://www.skillsoft.com/content-solutions/it-training-portfolio/software-development-training/>

<https://app.pluralsight.com/library/>

**Support**

After-project support. Solving software conflicts and usability problems, and supplying updates and patches for bugs and security holes in the program. During development the PM has to decide what resources to put forth after the deployment.

<https://www.techopedia.com/definition/21970/software-development-environment-sde>

<http://www.businessdictionary.com/definition/software-support.html>

**Service Levels (SLA’s)**

A contract between then client and service provider. It outlines aspects of the service to be provided - quality, availability, responsibilities

<https://www.cio.com/article/2438284/outsourcing/outsourcing-sla-definitions-and-solutions.html>

<http://www.infosysblogs.com/testing-services/2011/08/the_importance_of_service_leve.html>

**2.2 Explain the key technical concepts that affect the architecture of a software project, including but not limited to:**

* Deployment
* Environments
* Maintenance
* Decommissioning

**Deployment**

Deploy is a very loose term. Normally it refers to installing the code where it can be used. You can deploy released code, or code that is nowhere ready for release. In web development, it is common to deploy code to a test environment before testing it. I've created build chains that built code and deployed it automatically to a development server for developer testing. Many build chain tools include the capability to deploy successful builds.

<https://softwareengineering.stackexchange.com/questions/344184/what-is-the-difference-between-software-deployment-and-software-release>

<https://pdf.wondershare.com/business/software-deployment-process.html>

**Environments (SDE)**

A software development environment is an environment that automates or augments the routines involved in a software development cycle. This includes programming-in-the-many tasks like team and project management as well as programming-in-the-large tasks like configuration management.

<https://study.com/academy/lesson/environments-in-system-development-life-cycle.html>

**Maintenance**

Maintenance in software development is a continuous task before and after the program is released. This involves continuous work to make sure the software is working and updated. This can also include fixing the software if there are any bugs that are found by users.

<https://study.com/academy/lesson/maintenance-phase-in-sdlc.html>

<https://www.techwalla.com/articles/the-maintenance-phase-in-the-software-life-cycle>

**Decommissioning**

This is the destroying of a program. The decommissioning phase in development is when the software project is ending. This could be for many reasons, which include: Lack of funding, no longer being used, or out of date.

<http://www.enterpriseunifiedprocess.com/essays/retirementPhase.html>

<https://its.unl.edu/bestpractices/decommissioning-process-guide>

**2.3 Describe effective team-working and how this contributes to effective delivery of software projects, including but not limited to:**

* Organisational Behaviour theory
* Decision making
* Conflict resolution
* Collaboration
* Communication

**Organisation behaviour Theory**

Organisational behaviour is "the study of [human behavior](https://en.wikipedia.org/wiki/Human_behavior) in organizational settings, the interface between human behavior and the organization, and the organization itself"

Organization theory is concerned with explaining the workings of an organization as a whole or of many organizations. The focus of organizational theory is to understand the structure and processes of organizations and how organizations interact with each other and the larger society

<https://www.researchgate.net/profile/Arvind_Singh56/post/Hello_could_you_recommend_me_literature_of_organizational_psychologie/attachment/59d64e7179197b80779a7c7a/AS%3A492764661465088%401494495658927/download/The-impact-of-organizational-behaviour-on-employees-behaviour-in-Pharmaceutical-companies-in-selected-locations-of-Maharashtra-Viz_-Mumbai-Pune-Nasik-Sapna-Suri.pdf>

<https://en.wikipedia.org/wiki/Organizational_behavior>

**Decision Making**

Decision Making is the process or action for making important decisions. In software development, programmers are always having to make decisions on the spot, whether it be to include some code or to not include certain code, Decision making during development is an everyday occurrence, and if not carried out carefully, can lead to serious consequences.

<https://simpleprogrammer.com/making-decisions-simpler/>

<https://www.codeproject.com/Articles/1230088/Software-Project-Development-and-Decision-Making>

**Conflict Resolution**

Conflict resolution is a way for several people to find a solution to a disagreement. This can be the way to code a bit of the program, or it could be the design of the product. When a dispute arises, often the best course of action is negotiation to resolve the disagreement.

<https://adtmag.com/articles/2014/12/17/agile-conflict-resolution.aspx>

<https://softwareengineering.stackexchange.com/questions/19851/how-to-handle-conflicts-between-developers-in-your-team>

**Collaboration**

Collaborative Software Engineering: Concepts and Techniques. Collaboration is a central activity in software engineering, as all but the most trivial projects involve multiple engineers working together.

Software engineering projects are inherently cooperative, requiring many software engineers to coordinate their efforts to produce a large software system. Integral to this effort is developing shared understanding surrounding multiple artifacts, each artifact embodying its own model, over the entire development process.

Collaboration is working with someone or with multiple people to produce or create something. In the classic app-dev approach, professionals all too often [end up working against each other](https://searchsoftwarequality.techtarget.com/tip/Software-developers-and-testers-Cant-we-all-just-get-along) to create something

<https://sdtimes.com/collaboration-software-development-people-tools-run-show/>

<https://ieeexplore.ieee.org/document/993937>

<https://users.soe.ucsc.edu/~ejw/papers/whitehead-future-of-collaboration-in-se.pdf>

**Communication**

Communication in Distributed Software Development is an area of study that considers communication processes and their effects when applied to software development in a globally distributed development process. Communication can help in any different ways to get a program working. It means that there won’t be any problems. Problems that can occur without communication can be: two lots of code for the same thing, different design of the pages (eg, fonts, structure, and colour), writing the wrong code.

<http://www.agilemodeling.com/essays/communication.htm>

<https://www.hexacta.com/2016/11/10/communication-agile-software-development/>

**3. Developing software against a set of functional and non-functional requirements (15%, K2)**

In this topic, the apprentice will gain an understanding of the differences between functional and non-functional requirements. They will understand how to review requirements, assess their validity, testability and how they are used to drive software development activities.

The successful apprentice should be able to:

**3.1 Distinguish between a functional and a non-functional requirement.**

A functional requirement describes what a software system should do, while non-functional requirements place constraints on how the system will do so.

An example of a functional requirement would be:

A system must send an email whenever a certain condition is met (e.g. an order is placed, a customer signs up, etc).

A related non-functional requirement for the system may be:

Emails should be sent with a latency of no greater than 12 hours from such an activity.

<https://stackoverflow.com/questions/16475979/what-is-the-difference-between-functional-and-non-functional-requirement>

<https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

Functional Requirements

* Business Rules
* Transaction corrections, adjustments and cancellations
* Administrative functions
* Authentication
* Authorization levels
* Audit Tracking
* External Interfaces
* Certification Requirements
* Reporting Requirements
* Historical Data
* Legal or Regulatory Requirements

Non Functional Requirements

* Performance – for example Response Time, Throughput, Utilization, Static Volumetric
* Scalability
* Capacity
* Availability
* Reliability
* Recoverability
* Maintainability
* Serviceability
* Security
* Regulatory
* Manageability
* Environmental
* Data Integrity
* Usability
* Interoperability

**3.2 Identify the different types of non-functional requirements and the reasons they are important to the end-product of software development, including but not limited to:**

* Availability
* Capacity
* Performance
* Scalability
* Reliability
* Maintainability

**Availability**

Availability is gauged by the period of time that the system’s functionality and services are available for use with all operations. So, scheduled maintenance periods directly influence this parameter. And it’s important to define how the impact of maintenance can be minimized.

<https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/>

**Capacity**

The capacity is the amount of resources made available to the system, and scalability is the ability of the system to make use of these resources.

Capacity defines the ways in which the systems may be expected to scale-up by increasing hardware capacity based on the organization’s volume projections. For example, transactions per seconds, customers online, response time, and so on.

Capacity is delivering sufficient functionality required by the end users. A request for a web service to supply 2,000 requests per second, when that server is only capable of 200 requests per second, will not succeed. This is because the server is unable to handle the requested capacity.

<http://blog.soton.ac.uk/orion/analysis/nonfunctional-requirements-2/>

<https://www.oreilly.com/library/view/mastering-non-functional-requirements/9781788299237/deed840d-95b8-40eb-bd06-63a7da2d959a.xhtml>

**Performance**

The only way in which systems will meet their performance targets is for them to be specified clearly and unambiguously. It is a simple fact that if performance is not a stated criterion of the system requirements then the system designers will generally not consider performance issues. While loose or incorrectly defined performance specifications can lead to disputes between clients and suppliers. In many cases performance requirements are never ridged as system that does not fully meet its defined performance requirements may still be released as other consideration such as time to market.

<http://www.1202performance.com/atricles/how-to-write-performance-requirements-with-example/>

**Scalability**

Scalability is a non-functional property of a system that describes the ability to appropriately handle increasing (and decreasing) workloads. Sometimes, scalability is a requirement that necessitates the usage of a distributed system in the first place.

<http://berb.github.io/diploma-thesis/original/024_scalability.html#www>

**Reliability**

Software reliability testing is a field of software testing that relates to testing a software's ability to function, given environmental conditions, for a particular amount of time. Software reliability testing helps discover many problems in the software design and functionality.

This means that, if done correctly, there shouldn't be any problems with the software due to where or how it is being used.

<https://en.wikipedia.org/wiki/Software_reliability_testing>

<https://users.ece.cmu.edu/~koopman/des_s99/sw_reliability/>

**Maintainability**

Software maintainability is defined as the degree to which an application is repaired, or enhanced.

Software maintainability makes sure that the software you are building is able to be repaired, edited, and updated without having much issue. This is important because when you want to release a new version, you won't have to rewrite all the code just to allow one small feature.

Backing up previous versions of the work is very important and can help with maintenance because if there is an error when updating, you can go back to the old version which is saved

<https://www.castsoftware.com/glossary/software-maintainability>

<https://software.ac.uk/resources/guides/developing-maintainable-software>

**3.3 Recognise common ways in which software requirements can be expressed.**

There are several ways to express the software requirements of a program you are making. One of these ways can be shown in a specification sheet.

A specification sheet is a word document that tells the team and the user about the program that is being built. A couple of examples that would be on the spec sheet are:

* Purpose
* Features
* GUI
* Requirements

<https://www.e-education.psu.edu/geog468/sites/www.e-education.psu.edu.geog468/files/Geog_468_Project_Template.docx>

<https://hubtechinsider.wordpress.com/2011/07/28/how-do-you-write-software-requirements-what-are-software-requirements-what-is-a-software-requirement/>

<https://qracorp.com/write-clear-requirements-document/>

<http://www.pjsrivastava.com/a-short-guide-to-writing-software-requirements/>

**3.4 Describe the qualities of good requirements and the impact of poor requirements.**

Poor requirements lead to poor designs and tests and depending on when these issues are uncovered, development and testing rework is the result. This, along with the requirements revisions themselves, contribute to late product delivery.

A requirement needs to meet several criteria to be considered a “good requirement”. Good requirements should have the following characteristics:

* Unambiguous
* Testable (verifiable)
* Clear (concise, terse, simple, precise)
* Correct
* Understandable
* Feasible (realistic, possible)
* Independent
* Atomic
* Necessary
* Implementation-free (abstract)

For more information on each bullet point: <http://www.informit.com/articles/article.aspx?p=1152528&seqNum=4>

<https://blog.anvileight.com/posts/software-requirements-specifications-good-and-bad-examples/>

<https://www.blueprintsys.com/blog/poor_requirements_what_impact_do_they_have/>

**3.5 Explain how to determine the correct types of test coverage based on a requirement.**

Software testing is an essential activity in the software development and maintenance life cycles. It is a practice often used to decide and improve software quality.

Development is more systematic nowadays and organizations seek measures of testing completeness and effectiveness to show test completion criteria. Of them all, coverage is considered especially valuable.

Test coverage helps monitor the quality of testing, and assists testers to create tests that cover areas that are missing or not validated.

<https://www.softwaretestinghelp.com/test-coverage/>

**Software Development Testing**

**Functional testing types:**

* Unit testing
* Integration testing
* System testing
* Sanity testing
* Smoke testing
* Interface testing
* Regression testing
* Beta/Acceptance testing

**Non-Functional testing types include:**

* Performance Testing
* Load testing
* Stress testing
* Volume testing
* Security testing
* Compatibility testing
* Install testing
* Recovery testing
* Reliability testing
* Usability testing
* Compliance testing
* Localization testing
* Soak Testing
* Fuzz Testing

Software testing is used to make sure that the code is working as its meant to and there are no problems with the code.

If done successfully, it will remove all of the errors from the software, it insures that the performance of the software meets the requirements of the program.

You can test whilst you write your code. You can do this using the log and the debugging mode. This is known as Test Driven Development (TDD). You can also test in an iterative phase of development. This is a cycle of programming then testing, and going back and forth until the program works. You can also test before the deployment when the program is complete, then go back and make sure it all works.

**Black Box testing** is a technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. This is also called functional testing.

**White Box testing** is a technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

The many testing design tools are used to conduct the tests and help create test cases. There are many testing frameworks available to automate test which do all the testing for you. You can find open source frameworks online “common testing frameworks GNU”.

**Test driven development** is an evolutionary approach to developments which combines test-first development. The way you do this is to first write a test, then write the code to fulfill that test, then rewrite the code if necessary whilst still passing the test. This way of development is very slow, but it also means that you won’t get as many errors early on. This means that a lot of people would use this way of testing.

**Unit testing** the testing of an individual unit or group of related units. This falls under the class of white box testing. It is often done by the programmer to test that the unit they have developed is producing the expected output against the given input. This is more of an efficient way of testing because the programmer knows his code and can test and fix quicker.

**Integration testing** is testing in which a group of components are combined to produce an output. It is the interaction between software and hardware is tested in integration. This can fall under both white, and black box testing.

**Functional testing** is the testing to ensure that the specified functionality required in the system requirements work. This falls under the class of black box testing.

**System testing** is the testing to ensure that the software works in different environments (eg. operating systems, browsers, etc.). This is done with full system implementations and environments. This is black box testing.

**Stress testing** is testing to evaluate how systems behave under unfavorable conditions. This testing is conducted at beyond limits of the specifications to make sure it can handle. This is basically testing to try and break the program. This is black box testing.

**Penetration testing**, also known as pen testing, is a way of testing a computer system, network or web application to find vulnerabilities that an attacker could exploit. This makes sure that you won’t get any unwanted people getting into your system. This testing is very important and falls under black box testing.

**Performance testing** is the testing to assess the speed and effectiveness of the system to make sure it is generating results within a specified time defined on the requirements. This falls under black box testing.

**Usability testing** is performed to the perspective of the client to evaluate: how the GUI is user friendly, how easily the user can learn, after learning how proficiently can the client perform, how pleasing is it to use. The best way of doing this is releasing it to a small group of people to see how they find it. This falls under black box testing.

**Acceptance testing** is often done by the customer to ensure that the delivered product meets the requirements and works as the customer expected. Combined with **Usability testing**, this is **User Acceptance Testing (UAT)**. This is black box testing.

**Regression testing** is the testing after the modification of a system, component, or a group of relates unites to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. This type of testing would be a continuous test after new releases or updates. This is black box testing.

**Beta testing** is done by end users, a team outside development, or publicly releasing a fill pre-version of the product. The aim of beta testing is to cover unexpected errors. This is that last phase of testing before the full product is released. This falls under the class of black box testing.

**4. Implement software code following a logical approach (17.5%, K3)**

In this topic, the apprentice will learn and apply the fundamental principles and concepts of software development including abstraction, logic, algorithms and data representation. They will learn how to write code to solve problems and how to debug the resulting program.

The successful apprentice should be able to:

**4.1 Illustrate the fundamental concepts of programming; including but not limited to:**

* Procedural vs. Object-Oriented vs. Functional programming
* Compiled vs. Interpreted

**Procedural**

Procedural programming is a programming paradigm that uses a linear or top-down approach. It relies on procedures or subroutines to perform computations.

Procedural programming is also known as imperative programming.

<https://www.techopedia.com/definition/21481/procedural-programming>

**Object-Oriented**

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them.

**Functional Programming**

Functional programming (often abbreviated FP) is the process of building software by composing pure functions, avoiding shared state, mutable data, and side-effects. Functional programming is declarative rather than imperative, and application state flows through pure functions.

Functional programming is a declarative paradigm, meaning that the program logic is expressed without explicitly describing the flow control.

**Imperative** programs spend lines of code describing the specific steps used to achieve the desired results — the flow control: How to do things.

**Declarative** programs abstract the flow control process, and instead spend lines of code describing the data flow: What to do. The how gets abstracted away.

<https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0>

In most OOP languages, objects are instances of a class, and are seen as individual entities which interact with each other. These objects mimic the real world (to a certain degree). To highlight how different the approaches are in OOP and FP, I’ve borrowed this example below:

You run a company and you just decided to give all your employees a $10,000.00 raise. How would you tackle this situation programmatically?

In OOP:

1. Create Employee class which initializes with name and salary, has a change salary instance method

2. Create instances of employees

3. Use the each method to change salary attribute of employees by +10,000

In FP:

1. Create employees array, which is an array of arrays with name and corresponding salary

2. Create a change\_salary function which returns a copy of a single employee with the salary field updated

3. Create a change\_salaries function which maps through the employee array and delegates the calculation of the new salary to change\_salary

4. Use both methods to create a new dataset, named ‘happier employees’

We can see that the FP approach uses pure functions and adheres to immutability (note the use of map instead of each, where map returns a new altered dataset while each alters the attributes/state of the objects). With OOP, we cannot easily identify if the object has had the function called on it unless we start from the beginning and track if this has happened, whereas in FP, the object itself is now a new object with a different name, which makes it considerably easier to know what changes have been made.

**4.2 Demonstrate the core constructs used when writing code, including but not limited to:**

* Classes
* Objects
* Methods
* Variables
* Logic operators (e.g. AND, OR, NOT, NAND, NOR, XOR)

**Classes**

Classes are templates used in object-oriented programming to create objects, which are instances of that class. They are useful for organizing related variables and functions.

Take a circle, for example. A circle has a radius, a diameter, a circumference, and an area. You can create a class for circles, which serves as a template or cookie cutter of sorts for creating circle objects.

**Objects**

The first step in OOP is to identify all the objects the programmer wants to manipulate and how they relate to each other, an exercise often known as data modeling. Once an object has been identified, it is generalized as a class of objects (think of Plato's concept of the "ideal" chair that stands for all chairs) which defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. Objects communicate with well-defined interfaces called messages.

The concepts and rules used in object-oriented programming provide these important benefits:

* The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
* Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
* The definition of a class is reuseable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).
* The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

**Methods**

A method in object-oriented programming (OOP) is a procedure associated with a message and an object. An object is mostly made up of data and behavior, which form the interface that an object presents to the outside world. Data is represented as properties of the object and behavior as methods. For example, a Window object would have methods such as open and close, while its state (whether it is opened or closed) would be a property.

**Variables**

public class Employee{

//class properties

public string fname;

public string lname;

//class constructor function - runs automatically when a new instance created

public void Main(fname,lname){

//constructor used to initialise instance properties

this.fname=fname;

this.lname=lname;

}

//class method

public getName(){

return this.fname + ' ' + this.lname;

}

}

//create instance of Employee class

var emp1 = new Employee('Donald','Trump');

//invoke method of employee class

emp1.getName();

**Logic operators (e.g. AND, OR, NOT, NAND, NOR, XOR)**

A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels. The logic state of a terminal can, and generally does, change often, as the circuit processes data. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

<https://whatis.techtarget.com/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR>

**4.3 Recognise common structures and algorithms, including but not limited to:**

* Data structures
* Control structures
  + Iteration
  + Selection
  + Sequence
* Algorithms
  + Encryption
  + Searching
  + Sorting

**Data structures**

A data structure is a collection (grouping) of simple or structured data types and a set of rules (operations) for organizing and accessing the collection.

Some examples of data structures would be: an array, structs, classes, lists, strings, stacks, queues, files and tables. The underlying theme is that each structure has a defined organisation and a set of rules that implement and control the organisation.

**Control structures**

In a program, a control structure determines the order in which statements are executed.

**Iteration**

Iteration is the process where a set of instructions or statements is executed repeatedly for a specified number of time or until a condition is met

**WHILE**

*while ( expression )  
 statement*

**DO**

*do  
 statement  
 while ( expression ) ;*

**FOR**

*for ( expression-1(opt) ;  
 expression-2(opt) ; expression-3(opt))  
 statement*

**Selection**

The selection control structure is used for making decisions and branching statements.

**IF**

*if ( expression )  
 statement*

*else(opt)  
 else-statement(opt)*

**SWITCH**

*switch ( expression )  
 statement*

**Sequence**

Sequential statements like A := 3 are interpreted one after another, in the order in which they are written.

**Algorithms**

An algorithm is a procedure or formula for solving a problem, based on conducting a sequence of specified actions. A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

**Encryption**

An encryption algorithm is a component for electronic data transport security. Actual mathematical steps are taken and enlisted when developing algorithms for encryption purposes, and. varying block ciphers are used to encrypt electronic data or numbers.

**Searching**

A search algorithm is the step-by-step procedure used to locate specific data among a collection of data. It is considered a fundamental procedure in computing. In computer science, when searching for data, the difference between a fast application and a slower one often lies in the use of the proper search algorithm.

**Sorting**

In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order. ... Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists.

**4.4 Employ modularity and rational reuse of code, including but not limited to:**

* Design patterns
* Library functions
* Frameworks

**Design patterns**

Design Patterns are typical solutions to commonly occurring problems in software design. They are blueprints that can be taken and customized to solve a particular design problem in your code.

* [Command](http://gameprogrammingpatterns.com/command.html)
* [Flyweight](http://gameprogrammingpatterns.com/flyweight.html)
* [Observer](http://gameprogrammingpatterns.com/observer.html)
* [Prototype](http://gameprogrammingpatterns.com/prototype.html)
* [Singleton](http://gameprogrammingpatterns.com/singleton.html)
* [State](http://gameprogrammingpatterns.com/state.html)

Sequencing Patterns

* [Double Buffer](http://gameprogrammingpatterns.com/double-buffer.html)
* [Game Loop](http://gameprogrammingpatterns.com/game-loop.html)
* [Update Method](http://gameprogrammingpatterns.com/update-method.html)

Behavioural Patterns

* [Bytecode](http://gameprogrammingpatterns.com/bytecode.html)
* [Subclass Sandbox](http://gameprogrammingpatterns.com/subclass-sandbox.html)
* [Type Object](http://gameprogrammingpatterns.com/type-object.html)

Decoupling Patterns

* [Component](http://gameprogrammingpatterns.com/component.html)
* [Event Queue](http://gameprogrammingpatterns.com/event-queue.html)
* [Service Locator](http://gameprogrammingpatterns.com/service-locator.html)

Optimisation Patterns

* [Data Locality](http://gameprogrammingpatterns.com/data-locality.html)
* [Dirty Flag](http://gameprogrammingpatterns.com/dirty-flag.html)
* [Object Pool](http://gameprogrammingpatterns.com/object-pool.html)
* [Spatial Partition](http://gameprogrammingpatterns.com/spatial-partition.html)

<http://gameprogrammingpatterns.com/>

**Library functions**

In computer science, a library is a collection of non-volatile resources used by computer programs, often for software development. These may include configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, values or type specifications.

A software library generally consists of pre-written code, classes, procedures, scripts, configuration data and more. Typically, a developer might manually add a software library to a program to achieve more functionality or to automate a process without writing code for it. For example, when developing a mathematical program or application, a developer may add a mathematics software library to the program to eliminate the need for writing complex functions. All of the available functions within a software library can just be called/used within the program body without defining them explicitly. Similarly, a compiler might automatically add a related software library to a program on run time.

**Frameworks**

A framework is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful.

In computer systems, a framework is often a layered structure indicating what kind of programs can or should be built and how they would interrelate. Some computer system frameworks also include actual programs, specify programming interfaces, or offer programming tools for using the frameworks. A framework may be for a set of functions within a system and how they interrelate; the layers of an operating system; the layers of an application subsystem; how communication should be standardized at some level of a network; and so forth. A framework is generally more comprehensive than a protocol and more prescriptive than a structure.

**4.5 Show how to efficiently and effectively debug code, including but not limited to:**

* Types of error
* Exception handling
* Reproduction
* Logging
* Test Coverage

**Types of error**

Syntax

Logical

Run time error

Network error

**Exception handling**

Try, catch

Assert

**Reproduction**

Reproducing errors

**Logging**

Making sure that the errors have been noted down

Time of error

Type of error

Reason for error

User ID

**Test Coverage**

How much the application the test applies to.

Eg, Logging in phase, logging out phase, whole program

**SQL vs NoSQL: High-Level Differences**

SQL databases are primarily called as Relational Databases (RDBMS); whereas NoSQL database are primarily called as non-relational, distributed or document database.

SQL databases are tables based databases whereas NoSQL databases are document based, key-value pairs, graph databases or wide column store.

This means that SQL databases represent data in form of tables which consist of n number of rows of data whereas NoSQL databases are the collection of key-value pair, document, graph databases or wide-column stores which do not have standard schema definitions which needs adhered to.

SQL databases have predefined schema whereas NoSQL databases have dynamic schema for unstructured data.

Some SQL database example are MySQL, Oracle, Sqlite, Postgres and MS-SQL.

Some NoSQL databases examples are MongoDB, BigTables, Redis, RavenDb, Cassandra, Hbase, Neo4j, and CouchDb.

For complex queries, SQL databases are good for the complex query intensive environment whereas NoSQL databases are not good fit for complex queries.

For the type of data to be stored, SQL databases are not the best fut for hierarchical data storage. However, NoSQL databases fit better for the hierarchical data storage as it follows the key-value pair way of storing data similar to JSON data.

**5. Code against data sources (10%, K2)**

In this topic, the apprentice will learn to appreciate the importance of seamlessly connecting applications to permanent data storage. They will also understand types of data storage and their applications.

The successful apprentice should be able to:

**5.1 Illustrate the key differences between a variety of SQL and NoSQL databases, including but not limited to:**

* Relational
* Document
* Key-Value
* Graph
* Column store

**Relational**

Pros

o Relational databases work with structured data.

o They support ACID transactional consistency and support “joins.”

o They come with built-in data integrity and a large eco-system.

o Relationships in this system have constraints.

o There is limitless indexing. Strong SQL.

Cons

o Relational Databases do not scale out horizontally very well (concurrency and data size), only vertically, (unless you use sharding).

o Data is normalized, meaning lots of joins, which affects speed.

o They have problems working with semi-structured data.

http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/

**Document**

A document database is a type of non-relational database that is designed to store semistructured data as documents. Document databases are intuitive for developers to use because the data in the application tier is typically represented as a JSON document.

E.G. MongoDB.

**Key-Value**

A key-value database, or key-value store, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash table. Key-value databases work in a very different fashion from the better known relational databases (RDB).

Eg localStorage

**Graph**

A graph database is a database designed to treat the relationships between data as equally important to the data itself. It is intended to hold data without constricting it to a predefined model.

Top 8 Free Graph Databases

1.GraphDB Lite. GraphDB Lite is a free RDF triplestore that allows to store up to 100 million triples on a desktop computer. ...

2.Neo4j Community Edition. ...

3.OrientDB Community Edition. ...

4.Graph Engine. ...

5.HyperGraphDB. ...

6.MapGraph. ...

7.ArangoDB. ...

8.Titan.

**Column store**

A column store database is a type of database that stores data using a column oriented model. Column store databases are considered [NoSQL](https://database.guide/what-is-nosql/) databases, as they use a different data model to [relational databases](https://database.guide/what-is-an-rdbms/). Columns store databases use a concept called a keyspace. A keyspace is kind of like a [schema](https://database.guide/what-is-a-database-schema/) in the relational model. The keyspace contains all the column families (kind of like [tables](https://database.guide/what-is-a-table/) in the relational model), which contain rows, which contain columns.

Benefits of Column Store Databases

Some key benefits of columnar databases include:

* **Compression**. Column stores are very efficient at data compression and/or partitioning.
* **Aggregation queries**. Due to their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).
* **Scalability**. Columnar databases are very scalable. They are well suited to massively parallel processing ([MPP](https://database.guide/what-is-an-mpp-database/)), which involves having data spread across a large cluster of machines – often thousands of machines.
* **Fast to load and query**. Columnar stores can be loaded extremely fast. A billion row table could be loaded within a few seconds. You can start querying and analysing almost immediately.

**5.2 Explain in detail the concepts of a relational database.**

A relational database is a database that allows related data to be stored across multiple [tables](https://database.guide/what-is-a-table/), and linked by establishing a [relationship](https://database.guide/what-is-a-relationship/) between the tables. This provides an efficient way to store data, as you can enter data once, then reference it from elsewhere in the database.

Most modern RDBMSs provide the ability for you to create a relationship either programmatically, or via a graphical user interface (GUI). Using the GUI, you can see the relationship represented in a visual diagram.

There are 3 types of [relationships](https://database.guide/what-is-a-relationship/) in relational database design. They are:

* One-to-One
* One-to-Many (or Many-to-One)
* Many-to-Many

These are explained below.

**One-to-One**

A [row](https://database.guide/what-is-a-row/) in [table](https://database.guide/what-is-a-table/) A can have only one matching row in table B, and vice versa.

**One-to-Many (or Many-to-One)**

This is the most common relationship type. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A.

**Many-to-Many**

In a many-to-many relationship, a row in table A can have many matching rows in table B, and vice versa.

A many-to-many relationship could be thought of as two one-to-many relationships, linked by an intermediary table.

**5.3 Select appropriate SQL queries for commonly performed operations.**

Here is the list of commonly used excel operation.

* Select Data - SELECT \* FROM <table name>
* Sort Data - ORDER BY (ASC or DESC)
* Filter Data - WHERE <field name> = (value)
* Delete Records - DELETE
* Add Records - INSERT
* Update Data in Existing Record - UPDATE
* Show Unique Values - DISTINCT <field name> FROM
* Write an expression to generate new column - CREATE <column name>
* LookUp data from another table - SELECT \* FROM <table name1>, <table name2> WHERE <table name1>.ID=<table name2>.ID

**5.4 Summarise the importance of effective data modelling and normalisation.**

Database normalization is the process of organizing data into tables in such a way that the results of using the database are always unambiguous and as intended. Such normalization is intrinsic to relational database theory. It may have the effect of duplicating data within the database and often results in the creation of additional tables.

Database normalization’s ability to avoid or reduce data anomalies, data redundancies and data duplications, while improving data integrity, have made it an important part of the data developer's toolkit for many years. It has been one of the hallmarks of the relational data model

**6. Follow good coding practices (12.5%, K3)**

In this topic, the apprentice will learn that good coding practices aid the efficiency and quality of software development, and of the resulting end-products of development. They will also learn that there exists a range of open and organisational standards, where to source these and how to apply them.

The successful apprentice should be able to:

**6.1 Apply the correct fundamental software coding principles in different business contexts, including but not limited to:**

* Common principles (e.g. DRY)
* Defensive programming
* Commenting
* Refactoring
* Patterns / Anti-patterns

**Common principles (e.g. DRY)**

**Kiss** - Keep It Simple Stupid

**Dry** - Don’t Repeat Yourself

**Open / Closed** - aim to make your code open to extension but closed to modification.

**Composition > Inheritance** - objects with complex behaviors should do so by containing instances of objects with individual behaviors rather than inheriting a class and adding new behaviors.

**Single Responsibility** - every class or module in a program should only concern itself with providing one bit of specific functionality

**Separation of Concerns** - a program should be designed so that it has many different non-overlapping encapsulations, and these encapsulations shouldn’t know about each other.

**YAGNI** - You Aren’t Gonna Need It

**Defensive programming**

Defensive programming is a form of [defensive design](https://en.wikipedia.org/wiki/Defensive_design) intended to ensure the continuing function of a piece of [software](https://en.wikipedia.org/wiki/Software) under unforeseen circumstances. Defensive programming practices are often used where high availability, safety or security is needed

**Commenting**

A **comment** is a programmer-readable explanation or *annotation* in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters. The syntax of comments in various programming languages varies considerably.

Proper use of commenting can make code maintenance much easier, as well as helping make finding bugs faster. Further, commenting is very important when writing functions that other people will use.

* Use clear and understandable naming conventions
* Keep your comments meaningful and concise
* Remove commented out codes

**Refactoring**

Code refactoring is the process of restructuring existing computer code—changing the factoring—without changing its external behavior. Refactoring improves nonfunctional attributes of thesoftware

* Keep It Small

Refactoring is safest and cheapest when it is done in many small increments rather than in large batches. The worst extreme is the complete system re-write refactoring. The best refactoring activities take seconds or minutes to execute.

* Team Cohesion

Teamwork in Agile requires high levels of communication and collaboration. In refactoring work, teamwork applies just as much as in any other activity. Here, it is critical that all members of the team have a unified understanding of the principles and purpose of refactoring. But that is just the first level of team cohesion around refactoring.

**Patterns**

Software design pattern. In software development, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code.

**Structural** patterns generally deal with relationships between entities, making it easier for these entities to work together.

**Creational** patterns provide instantiation mechanisms, making it easier to create objects in a way that suits the situation.

**Behavioral** patterns are used in communications between entities and make it easier and more flexible for these entities to communicate.

**Anti-patterns**

A well formulated AntiPattern also tells you why the bad solution looks attractive (e.g. it actually works in some narrow context), why it turns out to be bad, and what positive patterns are applicable in its stead.

Development AntiPatterns utilize various formal and informal refactoring approaches. The following summaries provide an overview of the Development AntiPatterns found in this chapter and focus on the development AntiPattern problem. Included are descriptions of both development and mini-AntiPatterns. The refactored solutions appear in the appropriate AntiPattern templates that follow the summaries.

**The Blob**

Procedural-style design leads to one object with a lion’s share of the responsibilities, while most other objects only hold data or execute simple processes. The solution includes refactoring the design to distribute responsibilities more uniformly and isolating the effect of changes.

**[Continuous Obsolescence](https://sourcemaking.com/antipatterns/continuous-obsolescence)**

Technology is changing so rapidly that developers often have trouble keeping up with current versions of software and finding combinations of product releases that work together. Given that every commercial product line evolves through new releases, the situation is becoming more difficult for developers to cope with. Finding compatible releases of products that successfully interoperate is even harder.

**[Lava Flow](https://sourcemaking.com/antipatterns/lava-flow)**

Dead code and forgotten design information is frozen in an ever-changing design. This is analogous to a Lava Flow with hardening globules of rocky material. The refactored solution includes a configuration management process that eliminates dead code and evolves or refactors design toward increasing quality.

**[Ambiguous Viewpoint](https://sourcemaking.com/antipatterns/ambiguous-viewpoint)**

Object-oriented analysis and design (OOA&D) models are often presented without clarifying the viewpoint represented by the model. By default, OOA&D models denote an implementation viewpoint that is potentially the least useful. Mixed viewpoints don’t allow the fundamental separation of interfaces from implementation details, which is one of the primary benefits of the object-oriented paradigm.

**[Functional Decomposition](https://sourcemaking.com/antipatterns/functional-decomposition)**

This AntiPattern is the output of experienced, nonobject-oriented developers who design and implement an application in an object-oriented language. The resulting code resembles a structural language (Pascal, FORTRAN) in class structure. It can be incredibly complex as smart procedural developers devise very “clever” ways to replicate their time-tested methods in an object-oriented architecture.

**[Poltergeists](https://sourcemaking.com/antipatterns/poltergeists)**

Poltergeists are classes with very limited roles and effective life cycles. They often start processes for other objects. The refactored solution includes a reallocation of responsibilities to longer-lived objects that eliminate the Poltergeists.

**[Boat Anchor](https://sourcemaking.com/antipatterns/boat-anchor)**

A Boat Anchor is a piece of software or hardware that serves no useful purpose on the current project. Often, the Boat Anchor is a costly acquisition, which makes the purchase even more ironic.

**[Golden Hammer](https://sourcemaking.com/antipatterns/golden-hammer)**

A Golden Hammer is a familiar technology or concept applied obsessively to many software problems. The solution involves expanding the knowledge of developers through education, training, and book study groups to expose developers to alternative technologies and approaches.

**[Dead End](https://sourcemaking.com/antipatterns/dead-end)**

A Dead End is reached by modifying a reusable component if the modified component is no longer maintained and supported by the supplier. When these modifications are made, the support burden transfers to the application system developers and maintainers. Improvements in the reusable component are not easily integrated, and support problems can be blamed upon the modification.

**[Spaghetti Code](https://sourcemaking.com/antipatterns/spaghetti-code)**

Ad hoc software structure makes it difficult to extend and optimize code. Frequent code refactoring can improve software structure, support software maintenance, and enable iterative development.

**[Input Kludge](https://sourcemaking.com/antipatterns/input-kludge)**

Software that fails straightforward behavioral tests may be an example of an input kludge, which occurs when ad hoc algorithms are employed for handling program input.

**[Walking through a Minefield](https://sourcemaking.com/antipatterns/walking-through-minefield)**

Using today’s software technology is analogous to walking through a high-tech mine field. Numerous bugs are found in released software products; in fact, experts estimate that original source code contains two to five bugs per line of code.

**[Cut-and-Paste Programming](https://sourcemaking.com/antipatterns/cut-and-paste-programming)**

Code reused by copying source statements leads to significant maintenance problems. Alternative forms of reuse, including black-box reuse, reduce maintenance issues by having common source code, testing, and documentation.

**[Mushroom Management](https://sourcemaking.com/antipatterns/mushroom-management)**

In some architecture and management circles, there is an explicit policy to keep system developers isolated from the system’s end users. Requirements are passed second-hand through intermediaries, including architects, managers, or requirements analysts.

**6.2 Use open and organisational coding standards**

Coding standards are a set of guidelines, best practices, programming styles and conventions that developers adhere to when writing source code for a project. All big software companies have them

**6.3 Illustrate the importance of good coding standards.**

Coding standards can be understood as a series of procedures for a specific programming language that determines the programming style, procedures, methods, for various aspects of the program written in that language. They are a very critical attribute of software development.

A coding standard ensures that all developers writing the code in a particular language write according to the guidelines specified. This makes the code easy to understand and provides consistency in the code.

One of the most essential factors in a software system is the consistency of the coding standard. This is because it positively impacts the quality of the system. While using a software system, you need to ensure that the guidelines do not contradict each other. The source code that uses the standard should also be in harmony with the standard. The completed source code should indicate as if the code has been written by a single developer in a single session.

Without a coding standard, developers will use their own methods for coding and this has certain negative impacts.

**6.4 Recognise how automation can improve the overall quality and operational effectiveness of software product through processes including but not limited to;**

* Static analysis
* Code complexity
* Code coverage
* Continuous integration

**Static analysis**

<https://www.sealights.io/blog/benefits-of-automated-static-testing/>

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program. Analysis performed on executing programs is known as dynamic analysis. It can be argued that software metrics and reverse engineering are forms of static analysis.

The principal advantage of static analysis is the fact that it can reveal errors that do not manifest themselves until a disaster occurs weeks, months or years after release. Nevertheless, static analysis is only a first step in a comprehensive software quality-control regime. After static analysis has been done, dynamic analysis is often performed in an effort to uncover subtle defects or vulnerabilities. In computer terminology, static means fixed, while dynamic means capable of action and/or change.

**Code complexity**

How complex (how smart) a certain piece of code is.

Three complexity metrics for assessing code complexity.

**Cyclomatic Complexity.**

A quantitative measure of the number of linearly independent paths through a program’s source code…computed using the control flow graph of the program.

**Switch Statement and Logic Condition Complexity**  
the greater the number of nested conditions and the higher the level of complexity within those conditions, the higher the complexity of the code.

**Developer Skill**the skill level of the developer

<https://www.codacy.com/blog/an-in-depth-explanation-of-code-complexity/>

**Code coverage**

<https://www.sealights.io/test-metrics/code-coverage-vs-test-coverage-pros-and-cons/>

Code coverage is a measurement of how many lines/blocks/arcs of your code are executed while the automated tests are running.

Code coverage is collected by using a specialized tool to instrument the binaries to add tracing calls and run a full set of automated tests against the instrumented product. A good tool will give you not only the percentage of the code that is executed, but also will allow you to drill into the data and see exactly which lines of code were executed during a particular test.

**Continuous integration**

<https://www.infoworld.com/article/3295900/application-development/what-is-continuous-integration-ci-faster-better-software-development.html>

Continuous integration is a coding philosophy and related set of practices that drive development teams to implement small changes and check in code to version-control repositories frequently.

Because most modern applications require developing code in different platforms and tools, the team needs a mechanism to integrate and validate their changes. The technical goal of continuous integration is to establish a consistent and automated way to build, package, and test applications. With consistency in the integration process in place, teams are more likely to commit code changes more frequently, which leads to better collaboration and software quality.

**7. Understand the principles of good interface design (10%, K2)**

In this topic, the apprentice will learn the issues associated with designing and developing interactive systems. They will also appreciate the main techniques and technologies used for interface design and the importance of accessibility and usability when developing interactive systems.

The successful apprentice should be able to:

**7.1 Summarise the concept of Human Computer Interaction, it's history and goals.**

HCI (human-computer interaction) is the study of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings. A significant number of major corporations and academic institutions now study HCI. Historically and with some exceptions, computer system developers have not paid much attention to computer ease-of-use. Many computer users today would argue that computer makers are still not paying enough attention to making their products "user-friendly." However, computer system developers might argue that computers are extremely complex products to design and make and that the demand for the services that computers can provide has always outdriven the demand for ease-of-use.

User interface is one of the most important element of a software, which attract the user toward the application. Interaction refers to an abstract model by which humans interact with the computing device and interface is a choice of software .HCI refers to both interaction and interface. HCI give the idea of user experience UX in which we can improve the computing devices and application for the user. HCI is a design that should produce a good relation between the user, the device and the services which is performed by the device in order to achieve a certain task, both in quality and optimality of the services

HCI is an important factor when designing any of these systems or interfaces ATM machines, train ticket, hot drinks ,banking software ,management software ,aircraft and cars. Good use of HCI principles and techniques is not only important for the user which use your software, but it is very high priority for software development companies. Develop or improve Safety, Utility, Effectiveness, Efficiency, Usability and Appeal of systems that include computers.

**7.2 Describe the key concepts of good user interface design, including but not limited to:**

* Design principles
* Design patterns
* Tools (for example, Wireframes, Prototypes)
* Techniques (for example, Usability Testing, A/B testing)

**Design principles**

Design Principles are widely applicable laws, guidelines, biases and design considerations, all reflecting researchers’ and practitioners’ accumulated knowledge and experience. Design Principles draw from many disciplines—e.g., behavioral science, sociology, physics and ergonomics. Designers apply them with discretion.

Designers employ Design Principles such as visibility, findability and learnability to address basic human behaviors. We use some Design Principles, such as perceived affordances (e.g., buttons), to guide actions, putting users in control in seamless experiences.

Types of Principles:

* Keep users informed of system status with constant feedback.
* Set information in a logical, natural order.
* Ensure users can easily undo/redo actions.
* Maintain consistent standards so users know what to do next without having to learn new tool sets.
* Prevent errors if possible; wherever not, warn users before they commit to actions.
* Don’t make users remember information; keep options, etc. visible.
* Make systems flexible so novices and experts can choose to do more or less on them.
* Design with aesthetics and minimalism in mind – don’t clutter with unnecessary items.
* Provide plain-language error messages to pinpoint problems and likely solutions.
* Offer easy-to-search troubleshooting resources, if needed.
* Don’t interrupt or give users obstacles; make obvious pathways offering an easy ride.
* Offer few options – don’t hinder users with “nice to haves”; instead, give them needed alternatives.
* Reduce distractions – let users perform tasks consecutively, not simultaneously.
* Cluster related objects together.
* Have an easy-to-scan visual hierarchy mirroring users’ needs, with commonly used items handily available.
* Make things easy to find.
* Show users where they’ve come from and where they’re headed with signposts/cues.
* Provide context, showing how everything interconnects.
* Avoid jargon.
* Make designs efficient and streamlined.
* Use defaults wisely – offering predetermined, well-considered options helps minimize decisions and increase efficiency.
* Don’t delay users – ensure quick interface responses.
* Emotion – pleasure of use is as vital as ease of use; arousing users’ passion increases engagement.
* Less is more – make everything count in the design. If functional and aesthetic elements don’t add to the user experience, forget them.
* Be consistent with navigational mechanisms, organizational structure, etc., to make a stable, reliable and predictable design.
* Create a good first impression.
* Be trustworthy and credible – identify yourself via your design, ensuring users and eliminating uncertainty.

**Design patterns**

In the Human-Computer Interaction (HCI) community, design patterns are an often used tool for sharing design knowledge among user interface (UI) designers as well as non UI experts. An HCI design pattern consists of several different components.

**Problem**: Problems are related to the usage of the system and are relevant to the user or any other stakeholder that is interested in usability.

**Use when**: a situation (in terms of the tasks, the users and the context of use) giving rise to a usability problem. This section extends the plain problem-solutions dichotomy by describing situations in which the problems occur.

**Principle**: a pattern is usually based on one or more ergonomic principles such as user guidance, or consistency, or error management.

**Solution**: a proven solution to the problem. A solution describes only the core of the problem, and the designer has the freedom to implement it in many ways. Other patterns may be needed to solve sub problems.

**Why**: How and why the pattern actually works, including an analysis of how it may affect certain attributes of usability. The rationale (why) should provide a reasonable argument for the specified impact on usability when the pattern is applied. The why should describe which usability aspects should have been improved or which other aspects might suffer.

**Examples**: Each example shows how the pattern has been successfully applied in a real life system. This is often accompanied by a screenshot and a short description.

**Implementation**: Some patterns provide implementation details.

<https://www.interaction-design.org/literature/article/10-great-sites-for-ui-design-patterns>

**Tools (for example, Wireframes, Prototypes)**

**Wireframes (Low to Medium Fidelity)**  
A wireframe is a low fidelity representation of the product. It represents the layout of a webpage that demonstrates what interface elements will exist on key pages. It generally covers the 3 basic questions in the design process – What, Where & How.

* **What** are the main key elements of the layout? e.g. carousel, widget, left nav, hero space.
* **Where** are the elements going to be placed in the layout. e.g. top, bottom, left, right or middle section.
* **How** are they going to be placed.? e.g. in the form of a navigation, blocks of text, visual representation

**Prototypes (Medium to High Fidelity)**  
A prototype is a middle to high representation of the final product. It typically simulates the user interface interaction. A prototype allows the user to test the app for different interactions, which would be very similar to the interactions with the final product.

**Storyboards**  
A technique to visualise the user's journey or how to users would experience a problem or product.

**Techniques (for example, Usability Testing, A/B testing)**

**A/B testing**

Also known as split testing, is a method of comparing two versions of a digital product in order to find out which one performs better. A creative team divides users into two groups and each of them is shown different variants.

**Usability Testing**

The practice of testing how easy a design is to use on a group of representative users. It usually involves observing users as they attempt to complete tasks and can be done for different types of designs, from user interfaces to physical products. It is often conducted repeatedly, from early development until a product’s release.

**7.3 Discuss how the following platforms impact how an application is used and the changes required in Interface Design, including but not limited to:**

* Web
* Mobile
* Desktop

**Web**

New updates to different web browser impact what people can do on their websites. As websites are used by millions of people on a daily basis, every change that is meade will have a massive impact. This means that if you wanted to change the design on a new website, you would have to make sure that every website would be compatible.

In the field of Web design and development, we’re quickly getting to the point of being unable to keep up with the endless new resolutions and devices. For many websites, creating a website version for each resolution and new device would be impossible, or at least impractical.

Responsive Web design is the approach that suggests that design and development should respond to the user’s behavior and environment based on screen size, platform and orientation.

**Mobile**

**Small Screens**

You don’t have as much screen real estate for mobile devices as you do for PCs and laptops. That means, normally, you’ll be designing for multiple screen sizes. You need to make a decision early as to whether to use responsive design (where the device handles the changes in display) or adaptive design (where your servers handle the changes).

You want to focus on a “mobile first” approach which means designing for the smallest mobile platforms and increasing complexity from there.

**Navigation**

Keypads and touch screens don’t make for precise navigation like mice do

**Minimum Content**

Don’t overwhelm your users – respect the small screen space. Keep content to a minimum.

**Input from Users**

The less the user has to fiddle with their phone; the more they’re going to enjoy using your mobile web offering.

**Desktop**

Before designing a user interface, keep in mind, their will be a variety of users for your application such as expert users and novice users. Your UI should be effective enough that it doesn't seems to be too simple for expert users nor too complex for novice users. Design it in a way which can satisfy need of all audiences. Novice users move slower as they are uncertain about the functionality of elements of application whereas expert users move quickly throughout the application.

Simplicity is important in user interface design. The more controls you display on the screen at any time, the more time your users will have to spend figuring out how to use your interface. When there is less choice, the available functions become more apparent and are easier to scan. Simplifying an interface isn’t easy though, especially if you don’t want to limit the app’s functionality.

UI/UX should follow the right approach. Android, iOS, and Windows have design guidelines that comprise different design styles. When creating app’s UI/UX, develop it efficient enough for your expert users, and informative and simple enough for novice users.

**7.4 Summarise the fundamental considerations for developing an accessible system and the purpose of the Web Accessibility Initiative (WAI).**

When websites and web tools are properly designed and coded, people with disabilities can use them. However, currently many sites and tools are developed with accessibility barriers that make them difficult or impossible for some people to use.

Making the web accessible benefits individuals, businesses, and society. International web standards define what is needed for accessibility.

Web accessibility means that websites, tools, and technologies are designed and developed so that people with disabilities can use them. More specifically, people can:

* perceive, understand, navigate, and interact with the Web
* contribute to the Web

Web accessibility encompasses all disabilities that affect access to the Web, including:

* auditory
* cognitive
* neurological
* physical
* speech
* Visual

Web accessibility also benefits people without disabilities, for example:

* people using mobile phones, smart watches, smart TVs, and other devices with small screens, different input modes, etc.
* older people with changing abilities due to ageing
* people with “temporary disabilities” such as a broken arm or lost glasses
* people with “situational limitations” such as in bright sunlight or in an environment where they cannot listen to audio
* people using a slow Internet connection, or who have limited or expensive bandwidth

Accessible Rich Internet Applications (ARIA) is a set of attributes that define ways to make Web content and Web applications (especially those developed with JavaScript) more accessible to people with disabilities.

<div id="percent-loaded" role="progressbar" aria-valuenow="75"

aria-valuemin="0" aria-valuemax="100" />

**7.5 Outline the key factors of interface design relating to internationalization and Localisation.**

internationalization and localization are means of adapting computer software to different languages, regional differences and technical requirements of a target locale.Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes.

Internationalization typically entails:

* Designing and developing in a way that removes barriers to localization or international deployment. This includes such things as enabling the use of Unicode, or ensuring the proper handling of legacy character encodings where appropriate, taking care over the concatenation of strings, avoiding dependency in code of user-interface string values, etc.
* Providing support for features that may not be used until localization occurs. For example, adding markup in your DTD to support bidirectional text, or for identifying language. Or adding to CSS support for vertical text or other non-Latin typographic features.
* Enabling code to support local, regional, language, or culturally related preferences. Typically this involves incorporating predefined localization data and features derived from existing libraries or user preferences. Examples include date and time formats, local calendars, number formats and numeral systems, sorting and presentation of lists, handling of personal names and forms of address, etc.
* Separating localizable elements from source code or content, such that localized alternatives can be loaded or selected based on the user's international preferences as needed.

Localization is the process of adapting internationalized software for a specific region or language by translating text and adding locale-specific components. Localization (which is potentially performed multiple times, for different locales) uses the infrastructure or flexibility provided by internationalization (which is ideally performed only once, or as an integral part of ongoing development).

Often thought of only as a synonym for translation of the user interface and documentation, localization is often a substantially more complex issue. It can entail customization related to:

* Numeric, date and time formats
* Use of currency
* Keyboard usage
* Collation and sorting
* Symbols, icons and colors
* Text and graphics containing references to objects, actions or ideas which,in a given culture, may be subject to misinterpretation or viewed as insensitive.
* Varying legal requirements and many more things.

**8. Security in software development (17.5%, K2)**

In this topic, the apprentice will learn what the key approaches to securing a software system are and how they relate to the business. This includes the key steps in the design process to identify and incorporate security requirements into software development.

The successful apprentice should be able to:

**8.1 Describe the fundamentals of Information Security and Cyber Resilience, including but not limited to:**

* Scale and nature of risks
* Security versus Resilience
* Social engineering
* ISO/IEC 27001

**Scale and nature of risks**

Software development is an activity that uses a variety of technological advancements and requires high levels of knowledge. Because of these and other factors, every software development project contains elements of uncertainty. This is known as project risk. The success of a software development project depends quite heavily on the amount of risk that corresponds to each project activity. As a project manager, it’s not enough to merely be aware of the risks. To achieve a successful outcome, project leadership must identify, assess, prioritize, and manage all of the major risks.

The goal of most software development projects is to be distinctive—often through new features, more efficiency, or exploiting advancements in software development. Any software project executive will agree that the pursuit of such opportunities cannot move forward without risk.

**Security versus Resilience**

Security is defined as reducing the risk to critical infrastructure by physical means or defense cyber measures to intrusions, attacks, or the effects of natural or manmade disasters.

Examples of security measures:

* Badge entry at doors
* Using antivirus software
* Locking computer screens

Resilience is about the ability to prepare for and adapt to changing conditions and withstand and recover rapidly from disruptions. Resilience includes the ability to withstand and recover from deliberate attacks, accidents, or naturally occurring threats or incidents.

Examples of resilience measures:

* Developing a business continuity plan
* Having a generator for back-up power
* Using building materials that are more durable
* Having backups for if data is lost

**Social engineering**

Social engineering, in the context of information security, refers to psychological manipulation of people into performing actions or divulging confidential information.

The weakest link in the security chain is the human who accepts a person or scenario at face value. It doesn’t matter how many locks and deadbolts are on your doors and windows, or if have guard dogs, alarm systems, floodlights, fences with barbed wire, and armed security personnel; if you trust the person at the gate who says he is the pizza delivery guy and you let him in without first checking to see if he is legitimate you are completely exposed to whatever risk he represents.

**ISO/IEC 27001**

<https://www.iso.org/isoiec-27001-information-security.html>

The ISO/IEC 27000 family of standards helps organizations keep information assets secure.

Using this family of standards will help your organization manage the security of assets such as financial information, intellectual property, employee details or information entrusted to you by third parties.

ISO/IEC 27001 is the best-known standard in the family providing requirements for an information security management system (ISMS).

**What is an ISMS?**

An ISMS is a systematic approach to managing sensitive company information so that it remains secure. It includes people, processes and IT systems by applying a risk management process.

It can help small, medium and large businesses in any sector keep information assets secure.

**8.2 Outline proactive security approaches based on the following, including but not limited to:**

* Security development lifecycle (SDL)
* Defensive design (e.g. Defensive programming)
* Test creation and execution
* Permission setting and role based access
* Physical infrastructure and security

**Security development lifecycle (SDL)**

The Secure Development Lifecycle is a different way to build products; it places security front and center during the product or application development process. From requirements to design, coding to test, the SDL strives to build security into a product or application at every step in the development process.

In its simplest form, the SDL is a process that standardizes security best practices across a range of products and/or applications. It captures industry-standard security activities, packaging them so they may be easily implemented. The software development lifecycle consists of several phases:

In the **requirements** phase, best practices for security are integrated into a product. These practices may come from industry standards or be based on responses to problems that have occurred in the past.

The **design** phase of the SDL consists of activities that occur (hopefully) prior to writing code. Secure design is about quantifying an architecture (for a single feature or the entire product) and then searching for problems. Secure design could occur in a formal document or on a napkin.

The next phase is **implementation**, or writing secure code. The SDL contains a few things programmers must do to ensure that their code has the best chance of being secure. The process involves a mixture of standards and automated tools.

Formal **test** activities include security functional test plans, vulnerability scanning, and penetration testing. Vulnerability scanning uses industry-standard tools to determine if any system-level vulnerabilities exist with the application or product.

**Release** occurs when all the security activities are confirmed against the final build and the software is sent to customers (or made available for download). Response is the interface for external customers and security researchers to report security problems in products.

**Defensive design (e.g. Defensive programming)**

Defensive design is the practice of planning for contingencies in the design stage of a project or undertaking. ... Defensive design in software engineering is called defensive programming.

There are many definitions for Defensive Programming, it also depends on the level of “security” and level of resources you need for your software projects.

Defensive programming is a form of [defensive design](https://en.wikipedia.org/wiki/Defensive_design) intended to ensure the continuing function of a piece of [software](https://en.wikipedia.org/wiki/Software) under unforeseen circumstances. Defensive programming practices are often used where high availability, safety or security is needed

**Test creation and execution**

<https://www.atlassian.com/blog/archives/13-steps-to-learn-perfect-security-testing-in-your-org>

<https://dzone.com/articles/what-are-the-different-types-of-software-testing>

<https://insights.sei.cmu.edu/sei_blog/2018/07/10-types-of-application-security-testing-tools-when-and-how-to-use-them.html>

Security testing not only refers to testing the end product for security issues. It also ensures that plenty of proactive assurance techniques are being built in from the beginning of software development. A good security testing practice accounts for security assurance activities such as penetration testing, code review, and architecture analysis as integral elements of the development effort.

A security assessment normally starts by ensuring that the application includes the following attributes:

* Authentication
* Authorization
* Confidentiality
* Availability
* Integrity
* Non-repudiation
* Resilience

While security verification (i.e., testing) is an identified phase within the software development lifecycle (SDLC), it should be followed throughout the development process. Here's how to ensure your firm is including security throughout development and implementing critical attributes.

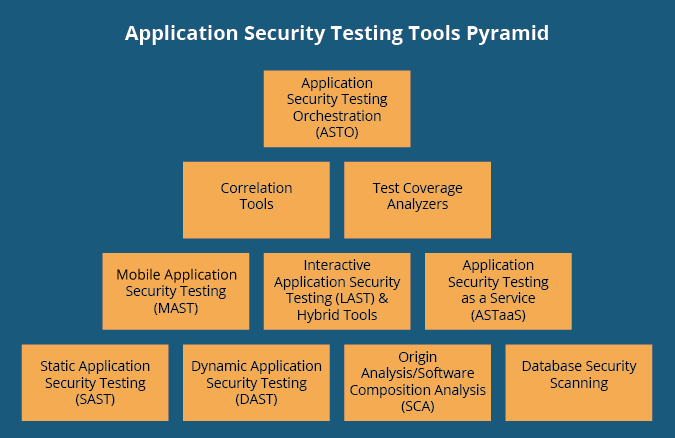
#### 

#### 

#### 

#### 

#### **10 Application Security Tools**



#### 

#### **Application Security Testing**

When an application is ready for quality and assurance testing, it's also ready for security testing. [Dynamic application security testing (DAST)](https://www.synopsys.com/software-integrity/security-testing/dynamic-analysis-dast.html) is a security scan that uses automated tools to identify common vulnerabilities within running web applications or web services - without the need for source code. This solution is ideal for internally-facing, low-risk applications that need to comply with regulatory security assessments. It can also be used for externally-facing applications; however, using DAST alone will not be sufficient.

Based on the type of application, organizations can also choose from the following manual penetration testing options. Each include client-side and server-side testing capabilities. These assessments can be white box (accompanied by source code), black box (testing without access to source code), or gray box (with some information - like configuration files - but without complete access to source code). Additionally, the duration and depth of analysis can be coordinated on a case by case basis.

Web application security penetration test. The application is written in one of the popular languages. Frameworks are tested for possible injection points and common vulnerabilities.

Mobile application penetration test. This includes the testing of applications written for the most popular mobile operating systems such as iOS, Android, Windows, and Blackberry.

Thick clients (desktop) application penetration test. Testing of the application written for desktop consumption.

A good commercial option is [Burp Scanner](http://portswigger.net/scanner); there are also free options such as OWASP’s [ZAP](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) and Google’s [RatProxy](http://code.google.com/p/ratproxy/). These work by routing the HTTP traffic to and from an application through a proxy, and then resending the requests with various attack attempts replacing the original values.

#### 

#### **Infrastructure Security Testing**

The infrastructure is often considered to be one of the most important aspects of maintaining software security. An unpatched piece of software risks exploitation. Leaking sensitive information can, as you probably well know, cause great monetary loss to a firm. Infrastructure testing assists the organization, ensuring that the network is equipped to withstand such issues through the following approaches:

* Network security penetration testing employs automated scanning and a manual testing checklist including test cases for encrypted transport protocols, SSL certificate scoping issues, use of administrative services, etc. Additionally, manual checks are conducted that are not normally found with automated testing. For example, vulnerabilities related to complex routing paths, access control configurations, business logic, and any functionality that is available through the exposed network services.
* Wireless penetrating testing. This engagement is carried out on the client-side with the assessor having access to the wireless network and covers configurations, wireless encryption standards, authentication, etc.
* Secure build of configuration review. This review ensures that the hosts have been properly hardened and patched. Permissions policies, password policies, and security settings are also tested. This can be included as a part of the network and wireless security test.
* [Red Teaming](https://www.synopsys.com/software-integrity/software-security-services/red-teaming.html). A combination of network, physical and social engineering techniques. It is used to assess an organization's security with the client's staff not being made aware of it. It also allows an organization to analyze its employees' security awareness and its own readiness against a real-world breach attempt.

**Permission setting and role based access**

**Physical infrastructure and security**

Infrastructure security is the security provided to protect infrastructure, especially critical infrastructure, such as airports, highways, rail transport, hospitals, bridges etc.

Critical infrastructure is vital for essential functioning of a country. Incidental or deliberate damage will have serious impact on the economy as well as providing essential services to the communities it serves. There are a number of reasons why infrastructure needs to be heavily secured and protected.

An example of one of the fundamental foundations of modern society is the electrical power systems. An intentional disruption of electricity supplies would affect national security, the economy, and every person's life. Because power grids and their sources are widely dispersed.

Security can be best achieved by ensuring multiple layers of security and not depending on a single measure. This principle is very evident here. The controls for physical and environmental security are defined in three areas:

* Security of the premise
* Security of the equipment
* Secure behavior

<http://www.networkmagazineindia.com/200302/security2.shtml>

**8.3 Summarise the key features of common security attacks; including but not limited to:**

* SQL injection
* Cross-site scripting
* Buffer overflows
* Malware
* (Distributed) Denial of Service
* Privilege escalation

**SQL injection**

SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious payload) that control a web application’s database server (also commonly referred to as a Relational Database Management System – RDBMS). Since an SQL Injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

<https://www.w3schools.com/sql/sql_injection.asp>

<https://www.acunetix.com/websitesecurity/sql-injection/>

**Cross-site scripting**

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

There are different types of Cross-Site Scripting:

**Stored XSS** generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser. With the advent of HTML5, and other browser technologies, we can envision the attack payload being permanently stored in the victim’s browser, such as an HTML5 database, and never being sent to the server at all.

**Reflected XSS** occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser (see DOM Based XSS next).

**DOM Based XSS** is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. For example, the source (where malicious data is read) could be the URL of the page (e.g., document.location.href), or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data (e.g., document.write)."

**Buffer overflows**

In information security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.

Buffer overflow is probably the best known form of software security vulnerability. Most software developers know what a buffer overflow vulnerability is, but buffer overflow attacks against both legacy and newly-developed applications are still quite common. Part of the problem is due to the wide variety of ways buffer overflows can occur, and part is due to the error-prone techniques often used to prevent them.

* This error occurs when there is more data in a buffer than it can handle, causing data to overflow into adjacent storage.
* This vulnerability can cause a system crash or, worse, create an entry point for a cyberattack.
* C and C++ are more susceptible to buffer overflow.
* Secure development practices should include regular testing to detect and fix buffer overflows. These practices include automatic protection at the language level and bounds-checking at run-time.
* CA Veracode binary SAST technology identifies code vulnerabilities, such as buffer overflow, in all code — including open source and third-party components —so that developers can quickly address them before they are exploited.

<https://www.veracode.com/security/buffer-overflow>

**Malware**

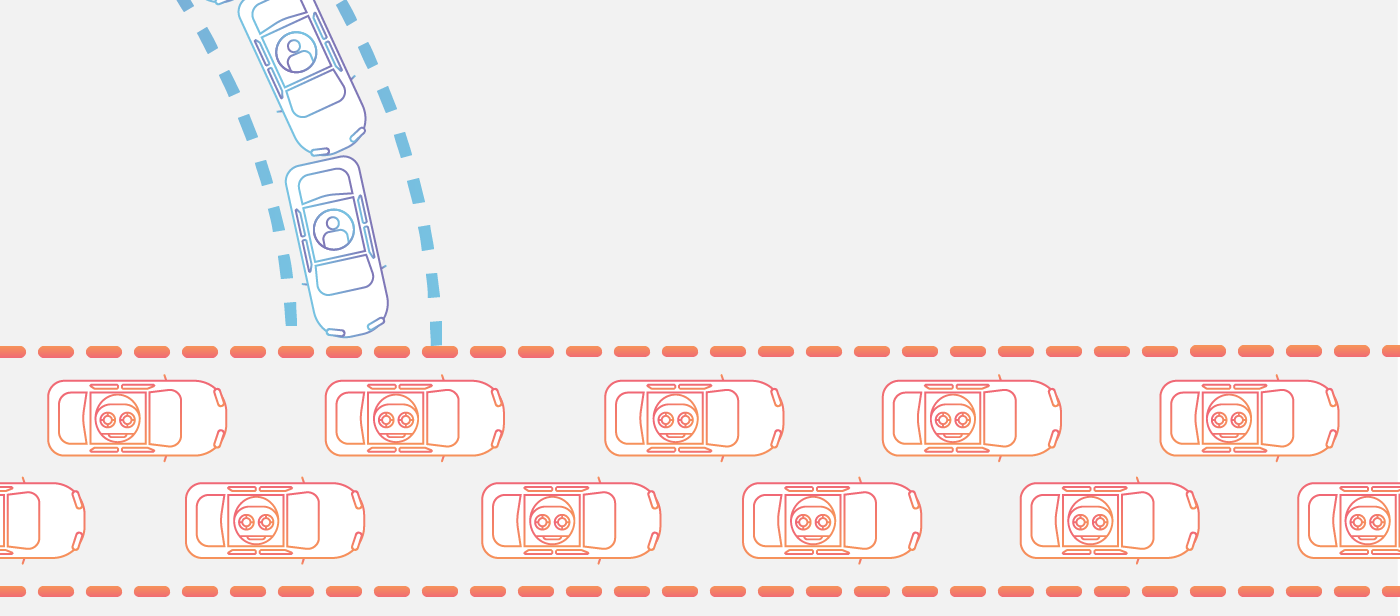
Malware is any software intentionally designed to cause damage to a computer, server or computer network. Malware does the damage after it is implanted or introduced in some way into a target’s computer and can take the form of executable code, scripts, active content, and other software.

Malware has a malicious intent, acting against the interest of the computer user—and so does not include software that causes unintentional harm due to some deficiency, which is typically described as a software bug.

A strategy for protecting against malware is to prevent the malware software from gaining access to the target computer. For this reason, antivirus software, firewalls and other strategies are used to help protect against the introduction of malware, in addition to checking for the presence of malware and malicious activity and recovering from attacks.

**(Distributed) Denial of Service**

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. DDoS attacks achieve effectiveness by utilizing multiple compromised computer systems as sources of attack traffic. Exploited machines can include computers and other networked resources such as IoT devices. From a high level, a DDoS attack is like a traffic jam clogging up with highway, preventing regular traffic from arriving at its desired destination.

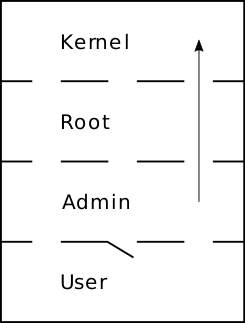
****

**DDoS ^**

[**https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/**](https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/)

**Privilege escalation**

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.

The arrow represents a rootkit gaining access to the kernel, and the little gate represents normal privilege elevation, where the user has to enter an Administrator username and password.

**8.4 Discuss typical approaches used to make available software more secure and manage business continuity; including but not limited to:**

* Security scanning
* Penetration testing
* Fuzzing
* Load testing

**Security scanning**

Security scanning or Vulnerability scanning is an inspection of the potential points of exploit on a computer or network to identify security holes. A vulnerability scan detects and classifies system weaknesses in computers, networks and communications equipment and predicts the effectiveness of countermeasures.

Running a vulnerability scan can pose its own risks as it is inherently intrusive on the target machine’s running code. As a result, the scan can cause issues such as errors and reboots, reducing productivity.

There are two approaches to vulnerability scanning, authenticated and unauthenticated scans. In the unauthenticated method, the tester performs the scan as an intruder would, without trusted access to the network. Such a scan reveals vulnerabilities that can be accessed without logging into the network. In an authenticated scan, the tester logs in as a network user, revealing the vulnerabilities that are accessible to a trusted user, or an intruder that has gained access as a trusted user.

**PENETRATION testing ;)**

**Penetration** testing is a systematic process of probing for vulnerabilities in your applications and networks. It is essentially a controlled form of hacking in which the ‘attackers’ operate on your behalf to find the sorts of weaknesses that criminals exploit.

The **penetration** testing process involves assessing your chosen systems for any potential weaknesses that could result from poor or improper system configuration, known and unknown hardware or software flaws, and operational weaknesses in process or technical countermeasures.

An experienced penetration tester can mimic the techniques used by criminals without causing damage. These tests are usually conducted outside business hours or when networks and applications are least used, thereby minimising the impact on everyday operations.

**Different types of penetration test**

* **External network (or infrastructure) penetration test**The objective of an external network penetration testing is to identify security vulnerabilities in how an organisation connects with the Internet and other external systems.
* **Internal network penetration test**The objective of an internal Network penetration test is to determine what vulnerabilities exist that are accessible to both an authenticated and non authenticated user to ensure that the network is critically assessed for both the potential exploit of a rogue internal user, and an unauthorised attack.
* **Web Application Penetration Testing**The objective of web application penetration testing is to identify security issues resulting from insecure development practices in the design, coding and publishing of software
* **Wireless network penetration test**The objective of wireless network penetration testing is to detect access points and rogue devices in an organisation’s secured environment.
* **Simulated phishing test**  
  The objective of phishing and social engineering penetration testing is to assess employees’ susceptibility to break security rules or give access to sensitive information.

<https://www.itgovernance.co.uk/penetration-testing>

<https://www.secureauth.com/products/penetration-testing>

**Fuzzing**

Fuzz testing (fuzzing) is a quality assurance technique used to discover coding errors and security loopholes in software, operating systems or networks. It involves inputting massive amounts of random data, called fuzz, to the test subject in an attempt to make it crash. If a vulnerability is found, a software tool called a fuzzer can be used to identify potential causes.

**Load testing**

Load testing involves applying ordinary stress to a software application or IT system to see if it can perform as intended under normal conditions. It is related to its bigger, more brutal cousin, *stress testing*, but load testing ensures that a given function, program, or system can simply handle what it’s designed to handle, whereas stress testing is about *overloading* things until they break, applying unrealistic or unlikely load scenarios. Both practices can play important roles in determining exactly how well a given piece of frontend software, such as a website, or a backend system, such as the Apache server hosting that site, can deal with the actual loads they’re likely to encounter through regular use.

**8.5 Identify common legal requirements within software including but not limited to:**

* Equality Act 2010
* EU Anti-Spam law
* EU Cookie law
* Patent Infringement and Copyright laws
* Privacy laws
* The Data Protection Act

**Equality Act 2010**

The Equality Act 2010 legally protects people from discrimination in the workplace and in wider society.

It replaced previous anti-discrimination laws with a single Act, making the law easier to understand and strengthening protection in some situations. It sets out the different ways in which it’s unlawful to treat someone.

Since 2 December 1996 (when the Disability Discrimination Act 1995 came into force) website owners have been obliged to ensure that their websites are accessible to users with disabilities. After over a decade in force, the DDA's requirements were merged into the Equality Act 2010. The 2010 Act was intended to bring clarity to the diversity of previously-extant discrimination legislation.

At the most basic ("priority 1") level of compliance, these include suggestions such as:

* Providing text to accompany non-text elements (such as pictures or graphical buttons for navigating).
* Document organisation for sensibly ordered readability without the need for the accompanying style sheets.
* Make sure all information conveyed through coloured content can be inferred or is available without colour.
* Clearly and simply labelling the websites content.
* Clearly delineating changes in the natural text of the document to other content, such as captions.

Compliance with both the priority 1 and 2 checklists is recommended. The priority 2 checklist includes:

* Ensuring the foreground and background colours have sufficient contrast for those who struggle with differentiating colours.
* Using an appropriate markup language rather than images to convey information.
* Using header elements to convey structure.
* Using style sheets to control the layout and presentation.
* Clearly identifying the target of each link.
* Providing further information about layout (e.g. a sitemap).
* Using navigation mechanisms in a consistent manner.
* Providing metadata to add semantic information to web pages.
* Dividing large blocks of information into more manageable blocks when possible.

**EU Anti-Spam law**

Anti-spam law restricts the sending of unsolicited marketing emails ('spam') to individual subscribers.

As a shop owner, you often have the opportunity to collect and use your customers email addresses. This can be a valuable resource to increase sales, but be sure to exercise caution when you are using this information.

The UK government has addressed privacy issues associated with this by implementing the Privacy and Electronic Communication Regulations in 2003. Compliance with this legislation is enforced by the Information Commissioner and carries significant monetary penalties.

**EU Cookie law**

The Cookie Law is a piece of privacy legislation that requires websites to get consent from visitors to store or retrieve any information on a computer, smartphone or tablet.

It was designed to protect online privacy, by making consumers aware of how information about them is collected and used online, and give them a choice to allow it or not.

It started as an EU Directive that was adopted by all EU countries in May 2011. The Directive gave individuals rights to refuse the use of cookies that reduce their online privacy. Each country then updated its own laws to comply. In the UK this meant an update to the Privacy and Electronic Communications Regulations.

Almost all websites use cookies to store information in people's' web browsers. Some websites contain hundreds of them.

There are other technologies, like Flash and HTML5 Local Storage that do similar things, and these are also covered by the legislation, but as cookies are the most common technology in use, it has become known as the Cookie Law.

**Patent Infringement and Copyright laws**

Patent infringement is the commission of a prohibited act with respect to a patented invention without permission from the [patent](https://en.wikipedia.org/wiki/Patent) holder. Permission may typically be granted in the form of a license. The definition of patent infringement may vary by jurisdiction, but it typically includes using or selling the patented invention. In many countries, a use is required to be *commercial* (or to have a *commercial* purpose) to constitute patent infringement

Infringement under United Kingdom patent law is defined by Section 60 of the UK Patents Act 1977 (as amended), which sets out the following types of infringement:

* Where the invention is a product, by the making, disposing of, offering to dispose of, using, importing or keeping a patented product.
* Where the invention is a process, by the use, or offer for use where it is known that the use of the process would be an infringement. Also, by the disposal of, offer to dispose of, use or import of a product obtained directly by means of that process, or the keeping of any such product whether for disposal or otherwise.
* By the supply, or offer to supply, in the United Kingdom, a person not entitled to work the invention, with any of the means, relating to an essential element of the invention, for putting the invention into effect, when it is known (or it is reasonable to expect such knowledge) that those means are suitable for putting, and are intended to put, the invention into effect in the United Kingdom.

Copyright applies to work that is recorded in some way; rights exist in items such as literary, artistic, musical and dramatic work as well as films, sound recordings and typographical arrangements. It gives the author specific rights in relation to the work, prohibits unauthorised actions, and allows the author to take legal action against instances of infringement or plagiarism.

<https://www.copyrightservice.co.uk/copyright/p01_uk_copyright_law>

**Data Privacy and Data Protection**

Data Privacy vs. Data Protection. ... In a nutshell, data protection is about securing data against unauthorized access. Data privacy is about authorized access — who has it and who defines it. Another way to look at it is this: data protection is essentially a technical issue, whereas data privacy is a legal one.

**Privacy laws**

Privacy laws in the U.K. seek to create a balance between your need to collect personal information and an individual's right to maintain their personal privacy. The law can be complex, and is based on eight data protection principles.

<https://www.loc.gov/law/help/online-privacy-law/2017/uk.php>

**The Data Protection Act**

The [Data Protection Act 2018](http://www.legislation.gov.uk/ukpga/2018/12/contents/enacted) controls how your personal information is used by organisations, businesses or the government.

The Data Protection Act 2018 is the UK’s implementation of the General Data Protection Regulation (GDPR).

Everyone responsible for using personal data has to follow strict rules called ‘data protection principles’. They must make sure the information is:

* used fairly, lawfully and transparently
* used for specified, explicit purposes
* used in a way that is adequate, relevant and limited to only what is necessary
* accurate and, where necessary, kept up to date
* kept for no longer than is necessary
* handled in a way that ensures appropriate security, including protection against unlawful or unauthorised processing, access, loss, destruction or damage

There is stronger legal protection for more sensitive information, such as:

* race
* ethnic background
* political opinions
* religious beliefs
* trade union membership
* genetics
* biometrics (where used for identification)
* health
* sex life or orientation

There are separate safeguards for personal data relating to criminal convictions and offences.