BSc in Computer Engineering
CMP4103
Computer Systems and Network Security

Lecture 7

Secure Software Development

Eng Diarmuid O'Briain, CEng, CISSP

Department of Electrical and Computer Engineering,
College of Engineering, Design, Art and Technology,
Makerere University

Table of Contents

This page is intentionally blank

1. Software Development Controls

1.1 Software Assurance

Software Assurance (SwA) is defined as '*the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle and that the software functions in the intended manner*'.

1.2 Avoiding System Failure

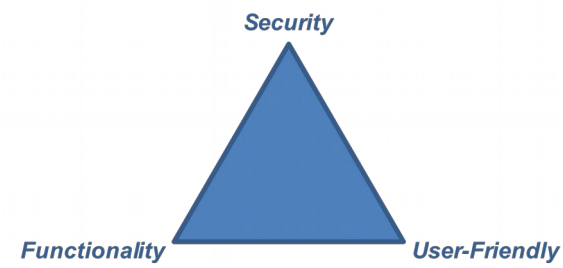
1.2.1 Fail-secure

Fail-secure describes a device or feature which, in the event of failure, responds in a way that will put the system into the highest level of security until the problem is diagnosed and repaired. For example, a fail-secure lock will remain locked during a failure, but cannot be unlocked even by the correct key.

1.2.2 Fail-open

Fail-open describes a device or feature which, in the event of failure allows users to bypass failed security controls, assuming those that want access should have access.

1.3 Programming Languages



A programming language is an artificial language designed to express computations that can be performed by a machine, particularly a computer. Programming languages can be used to create programs that control the behaviour of a machine, to express algorithms precisely, or as a mode of human communication.

Many programming languages have some form of written specification of their syntax (form) and semantics (meaning). Some languages are defined by a specification document. For example, the C programming language is specified by an ISO Standard. Other languages, such as Perl, have a dominant implementation that is used as a reference.

The earliest programming languages pre-date the invention of the computer, and were used to direct the behaviour of machines such as Jacquard looms and player pianos. Thousands of different programming languages have been created, mainly in the computer field, with many more being created every year. Most programming languages describe computation in an imperative style, i.e., as a sequence of commands, although some languages, such as those that support functional programming or logic programming, use alternative forms of description.

1.3.1 Programming Language Generations

- **First Generation Languages (1GL)**
  - Machine-level programming language.
  - Originally, no translator was used to compile or assemble the first-generation language. The first-generation programming instructions were entered through the front panel switches of the computer system.
- **Second Generation Languages (2GL)**
  - Assembly languages
  - The term was coined to provide a distinction from higher level third-generation programming languages (3GL) such as COBOL and earlier machine code languages.
- **Third Generation Languages (3GL)**
  - First introduced in the late 1950s, Fortran, ALGOL and COBOL are early examples of this sort of language.
  - Most "modern" languages (BASIC, C, C++, C#, Pascal, and Java) are also third-generation languages.
- **Fourth Generation Languages (4GL)**
  - High-level computer language such as Structured Query Language (SQL) that allows non-programmer users to write (usually short) programs to query databases and to generate custom reports.
- **Fifth Generation Languages (5GL)**
  - Solve problems using constraints given to the program, rather than using an algorithm written by a programmer.
  - Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.
  - This generation allows programmers to create code using visual interfaces.

1.3.2 Object-oriented Programming (OOP)

Example: C++, Java

OOP is a programming paradigm that uses "*objects*", data structures consisting of datafields and methods together with their interactions, to design applications and computer programs. Programming techniques may include features such as information hiding, data abstraction, encapsulation, modularity, polymorphism, and inheritance. It was not commonly used in mainstream software application development until the early 1990s.

An OOP may thus be viewed as a collection of interacting objects, as opposed to the conventional model, in which a program is seen as a list of tasks (subroutines) to perform. In OOP, each object is capable of receiving messages, processing data, and sending messages to other objects and can be viewed as an independent 'machine' with a distinct role or responsibility. The actions (or "operators") on these objects are closely associated with the object. For example, the data structures tend to 'carry their own operators around with them' (or at least "*inherit*" them from a similar object or class).

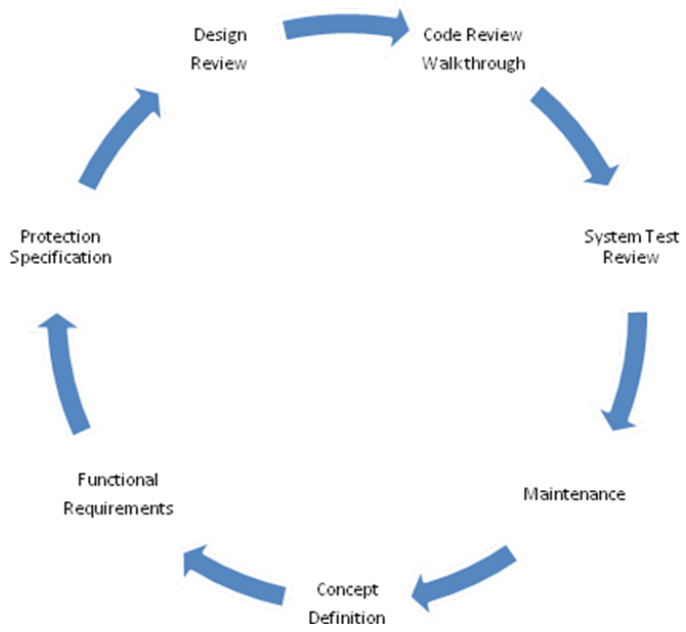
OOP Terms

- **Class**
  - Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields or properties) and the thing's behaviours (the things it can do, or methods, operations or features). i.e. the class Cat would consist of traits shared by all cats, such as breed and characteristics (coat colour), and behaviours.
- **Object**
  - A pattern (exemplar) of a class. The class of Cat defines all possible cats by listing the characteristics and behaviours they can have, the object Tom is one particular cat, with particular versions of the characteristics. A Cat has a coat. Tom has brown coat.
- **Instance**
  - One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Tom object is an instance of the Cat class. The set of values of the attributes of a particular object is called its state. The object consists of state and the behaviour that's defined in the object's class.

- **Method**
  - An object's abilities. In language, methods (sometimes referred to as "*functions*") are verbs. Tom, being a cat, has the ability to purr. So purr() is one of Tom's methods. She may have other methods as well, for example sit() or eat() or walk() or groom(). Within the program, using a method usually affects only one particular object, all cats can purr, but you need only one particular cat to do the purring.
- **Message passing**
  - "*The process by which an object sends data to another object or asks the other object to invoke a method.*" Also known to some programming languages as interfacing. For example, the object called Breeder may tell the Tom object to sit by passing a "*sit*" message which invokes Tom's "*sit*" method.
- **Inheritance**
  - "*Subclasses*" are more specialised versions of a class, which inherit attributes and behaviours from their parent classes, and can introduce their own.
  - i.e. the class Cat might have sub-classes called Tabby, White, and ColourPoint. In this case, Tom would be an instance of the Tabby subclass. Suppose the Cat class defines a method called purr() and a property called coatColor. Each of its sub-classes (Tabby, White, and ColourPoint) will inherit these members, meaning that the programmer only needs to write the code for them once.
- **Abstraction**
  - Abstraction is simplifying complex reality by modelling classes appropriate to the problem, and working at the most appropriate level of inheritance for a given aspect of the problem.
  - i.e. Tom the Cat may be treated as a Cat much of the time, a Tabby when necessary to access Tabby specific attributes or behaviours, and as an Animal (perhaps the parent class of Cat) when counting the pets.

- **Encapsulation**
  - Encapsulation conceals the functional details of a class from objects that send messages to it.
  - i.e. the Cat class has a purr() method. The code for the purr() method defines exactly how a purr happens (e.g., by inhale() and then exhale(), at a particular pitch and volume).
- **Polymorphism**
  - Polymorphism allows the programmer to treat derived class members just like their parent class' members. More precisely, Polymorphism in OOP is the ability of objects belonging to different data types to respond to method calls of methods of the same name, each one according to an appropriate type-specific behaviour.
  - i.e. If a Cat is commanded to speak(), this may elicit a purr(). However, if a Dog is commanded to speak(), this may elicit a bark(). They both inherit speak() from Animal, but their derived class methods override the methods of the parent class; this is Overriding Polymorphism.
- **Decoupling**
  - Decoupling allows for the separation of object interactions from classes and inheritance into distinct layers of abstraction. A common use of decoupling is to polymorphically decouple the encapsulation, which is the practice of using reusable code to prevent discrete code modules from interacting with each other. However, in practice decoupling often involves trade-offs with regard to which patterns of change to favour. The science of measuring these trade-offs in respect to actual change in an objective way is still in its infancy.

1.4 Systems Development Life Cycle (SDLC)



The SDLC in systems engineering and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems.

In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system: the software development process.

1.4.1 Concept Definition

Creation of a basic concept statement for the system. The Stakeholders must agree this before moving to the next phase.

1.4.2 Functional Requirements

The Functional Requirements Definition is put together by the development team. From this document the development team will design the elements of the system. At the end of the project this document will be used by Project Managers to ensure the functionality is met by the system.

1.4.3 Protection Specification

This is a specification of the security requirements of the system and how they will be developed within the system development.

1.4.4 Design Review

Once the Functional Specification and the Protection Specification have been complete the System Designers determine how the elements of the system will interoperate and communicate. The output of this stage is a Design Document.

1.4.5 Code Review Walk through

This should be a series of software code reviews as various stages of the development of code complete. This is managed by the development Project Manager (PM).

1.4.6 System Test Review

This is the period of intense testing by the Systems Test Group. The working code is sent to beta sites to test in near live environments and the discovered bugs are recorded.

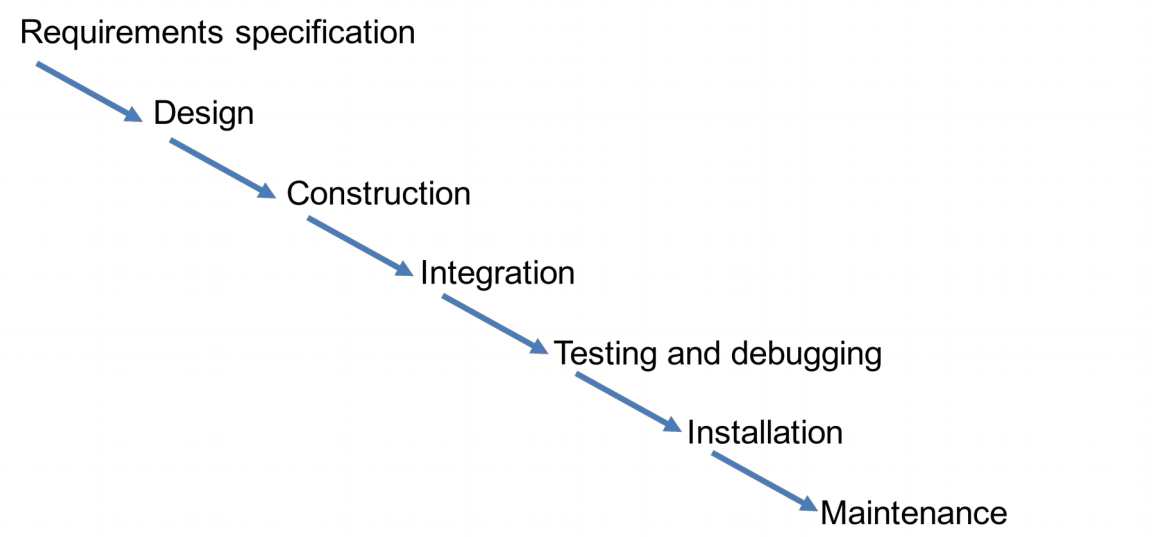
1.4.7 Maintenance

Once the product is released a maintenance cycle is necessary to keep the product operational as expected. Maintenance code releases with bug fixes will become necessary from time to time.

1.5 Life Cycle Models

1.5.1 Waterfall Model

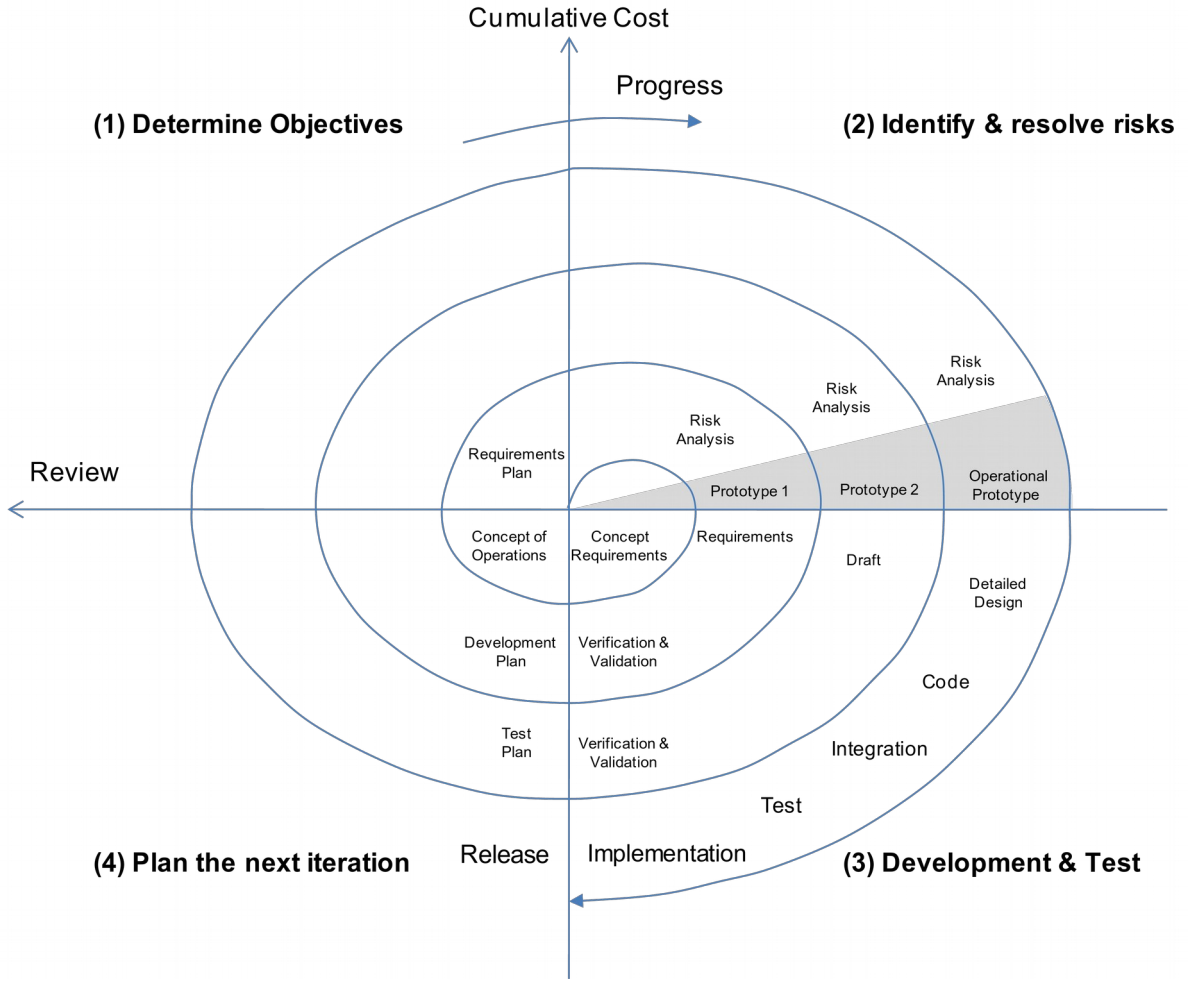
The waterfall model shows a process, where developers are to follow these steps in order:



After each step is finished, the process proceeds to the next step, just as builders don't revise the foundation of a house after the framing has been erected.

There is a misconception that the process has no provision for correcting errors in early steps (for example, in the requirements). In fact this is where the domain of requirements management comes in, which includes change control. The counter argument, by critics to the process, is the significantly increased cost in correcting problems through introduction of iterations. This is also the factor that extends delivery time and makes this process increasingly unpopular even in high risk projects.

1.5.2 Spiral Model



The spiral model is a software development process combining elements of both design and prototyping in stages, in an effort to combine advantages of top-down and bottom up concepts. Also known as the spiral lifecycle model combines the features of the prototyping model and the waterfall model. The spiral model is intended for large, expensive and complicated projects.

The steps in the spiral model iteration can be generalised as follows:

1) The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.

2) A preliminary design is created for the new system. This phase is the most important part of "Spiral Model". In this phase all possible (and available) alternatives, which can help in developing a cost effective project are analysed and strategies are decided to use them. This phase has been added specially in order to identify and resolve all the possible risks in the project development. If risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed with the available data and find out possible solution in order to deal with the potential changes in the requirements.

3) A first prototype of the new system is constructed from the preliminary design. This is usually a scaled down system, and represents an approximation of the characteristics of the final product.

4) A second prototype is evolved by a fourfold procedure:

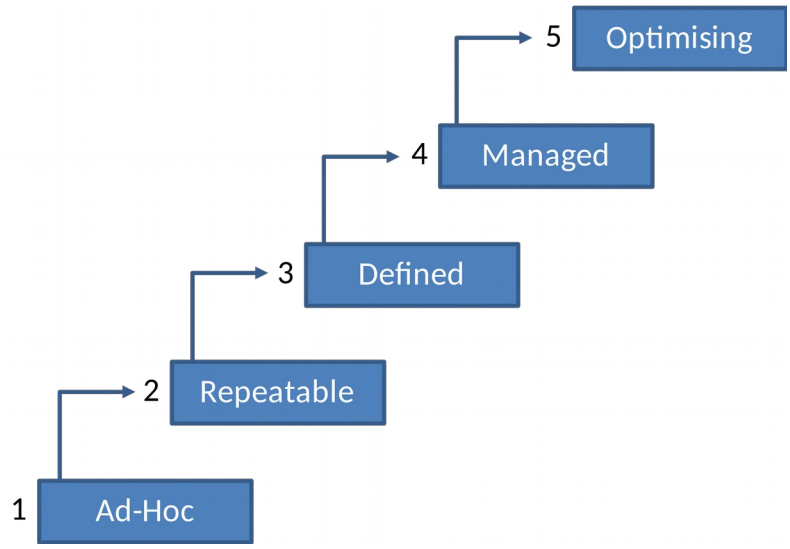
    a) Evaluating the first prototype in terms of its strengths, weaknesses, and risks

    b) Defining the requirements of the second prototype

    c) Planning and designing the second prototype

    d) Constructing and testing the second prototype.

1.5.3 Software Capability Maturity Model

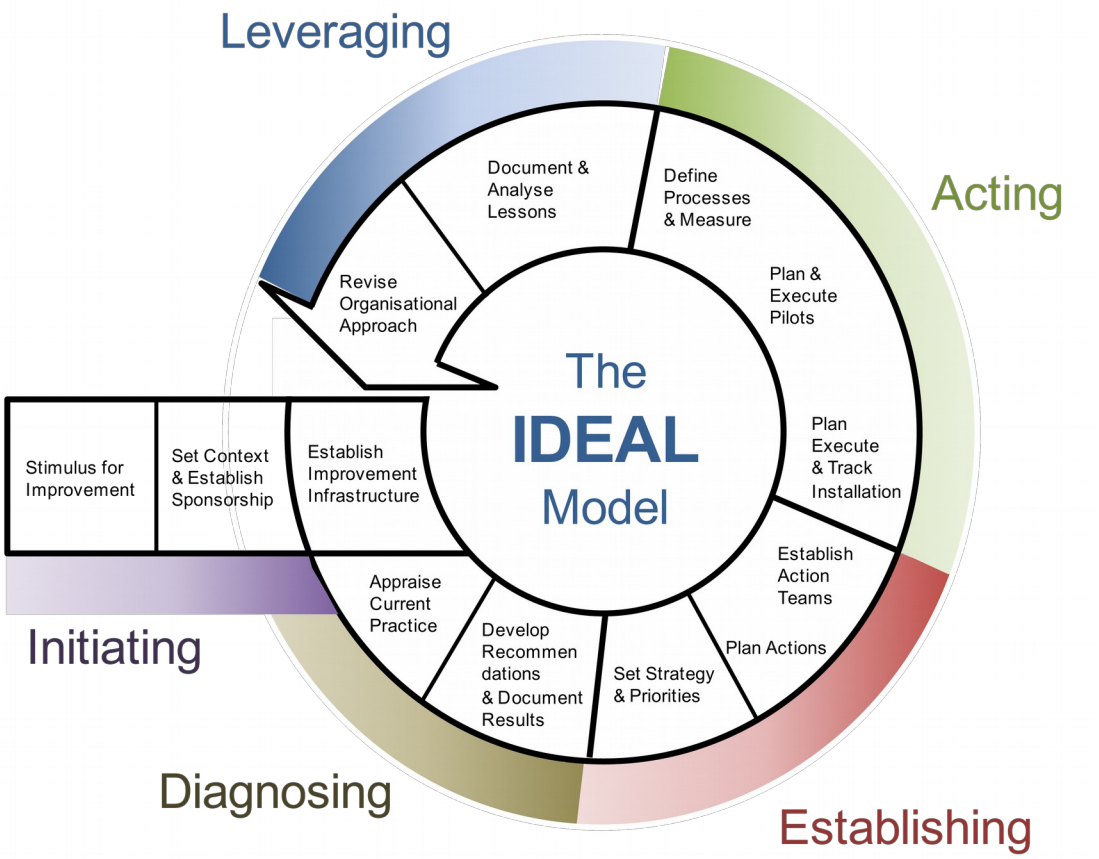


The Capability Maturity Model (CMM) is a service mark and a model for understanding the capability maturity of an organisation's software development business processes. Because the CMM is about process maturity, it differs from more common maturity models that provide a structured collection of elements that describe certain aspects of maturity in an organisation. The CMM is useful as a general theoretical model, to aid in the definition and understanding of an organisation's process capability maturity. For software development, the CMM has been superseded by Capability Maturity Model Integration (CMMI).

- **Level 1 - Initial**
  - It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an ad-hoc, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes.
- **Level 2 - Repeatable**
  - It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results.
  - Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

- **Level 3 - Defined**
  - It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place and are used to establish consistency of process performance across the organisation.
- **Level 4 - Managed**
  - It is characteristic of processes at this level that, using process metrics, management can effectively control the process. In particular, management can identify ways to adjust and adapt the process without measurable loss of quality or deviations from specifications. Process Capability is established from this level.
- **Level 5 - Optimised**
  - It is characteristic of processes at this level that the focus is on continually improving process performance through both incremental and innovative technological changes/improvements.

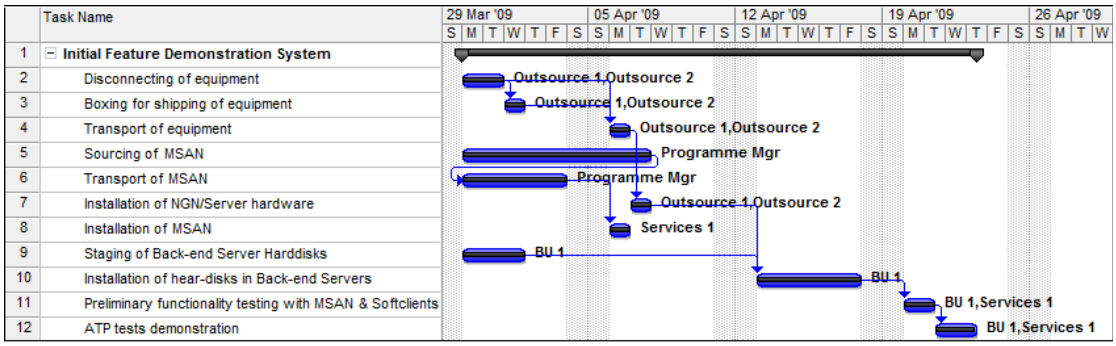
1.5.4 IDEAL model



The IDEAL model is an organisational improvement model that serves as a roadmap for initiating, planning, and implementing improvement actions. The IDEAL model is named for the five phases it describes:

- Initiating
- Diagnosing
- Establishing
- Acting
- Leveraging.

1.6 Gantt Chart



A Gantt chart is a type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent complete shadings and a vertical "TODAY" line as shown here.

1.7 Program Evaluation and Review Technique (PERT)

PERT is a method to analyse the involved tasks in completing a given project, especially the time needed to complete each task, and identifying the minimum time needed to complete the total project.

PERT was developed primarily to simplify the planning and scheduling of large and complex projects. It was able to incorporate uncertainty by making it possible to schedule a project while not knowing precisely the details and durations of all the activities. It is more of an event-oriented technique rather than start and completion oriented, and is used more in projects where time, rather than cost, is the major factor. It is applied to very large-scale, one-time, complex, non-routine infrastructure and Research and Development projects.

This project model was the first of its kind, a revival for scientific management, founded by Frederick Taylor "Taylorism" and later refined by Henry Ford "Fordism". DuPont Corporation's critical path method was invented at roughly the same time as PERT.

1.7.1 Determine the Critical Path

The Critical Path is the longest necessary path through a network of activities when respecting their interdependencies.

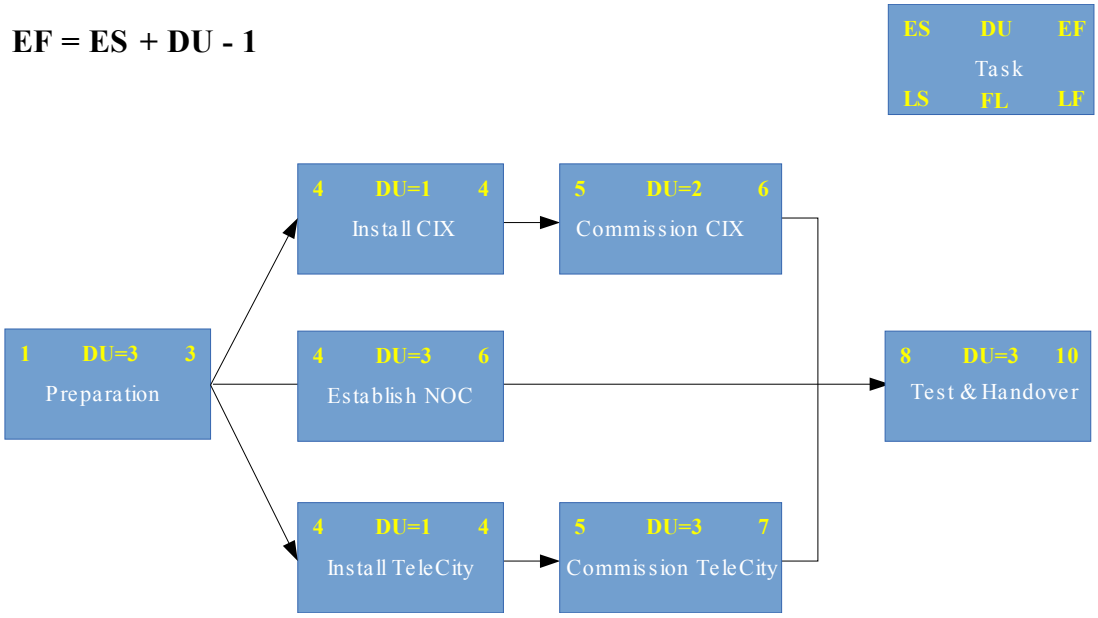
There are two terms related to the Critical Path. These are the terms Forward Pass and Backward Pass. These terms are related to ways of determining the early or late start [forward pass] or early or late finish [backward pass] for an activity.

Forward pass is a technique to move forward through a diagram to calculate activity duration. Backward pass is its opposite.

To determine the paths use a Program Evaluation and Review Technique (PERT) tool to layout the activity steps.

1.7.2 Forward pass calculations

$$EF = ES + DU - 1$$

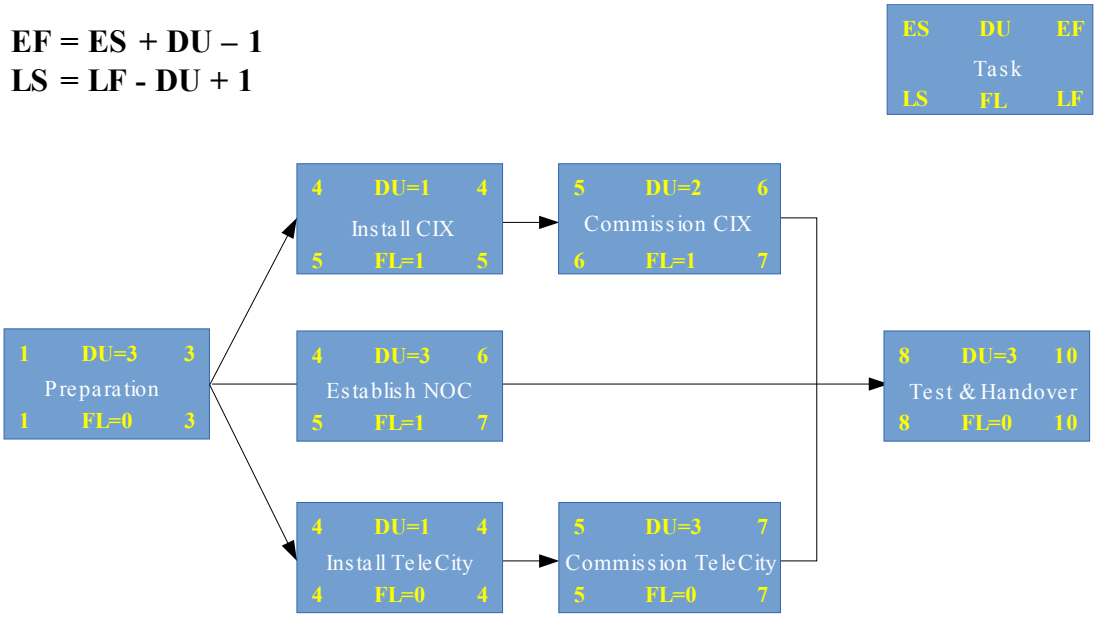


- **Early Start Date (ES)**
  - Earliest possible point in time an activity can start, based on the network logic and any schedule constraints.
- **Duration (DU)**
  - Number of work periods, excluding holidays or other non-working periods, required to complete the activity, expressed as workdays or workweeks.
- **Early Finish Date (EF)**
  - Earliest possible time the activity can finish.

Starting at the beginning (left) of the network develop ES and EF dates for each task, progressing to end (right-most box) of the network where EF = ES + DU – 1.

1.7.3 Backward pass calculations

$$EF = ES + DU - 1$$
$$LS = LF - DU + 1$$



- **Late Start Date (LS)**
  - Latest point in time that an activity may begin without delaying that activity's successor.
  - If the activity is on the critical path, the project end date will be affected.
- **Float or Slack (FL)**
  - Latest point in time a task may be delayed from its earliest start date without delaying the project finish date.
- **Late Finish (LF)**
  - Latest point in time a task may be completed without delaying that activity's successor If the activity is on the critical path, the project end date will be affected.

Calculate LS and LF dates by starting at project completion, using finish times and working backwards.

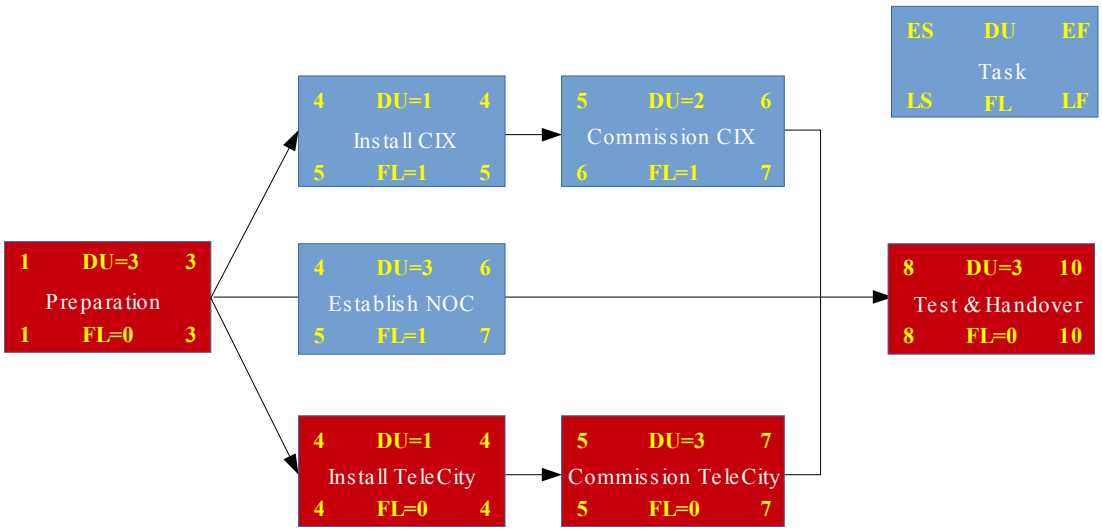
1.7.4 Forward and backward pass summary table

Forward and backward pass calculation						
Name	**Duration**	**ES**	**EF**	**LS**	**LF**	**FL**
Preparation	3	1	3	1	3	0
Install CIX	1	4	4	5	5	1
Establish NOC	3	4	6	5	7	1
Install TeleCity	1	4	4	4	4	0
Commission CIX	2	5	6	6	7	1
Commission TeleCity	3	5	7	5	7	0
Test & Handover	3	8	10	8	10	0

1.7.5 Calculating the Critical Path

The Critical Path is the longest possible continuous pathway taken from the initial event to the terminal event.

It determines the total calendar time required for the project. Therefore, any time delays along the critical path will delay the reaching of the final event by at least the same amount.



1.8 Change Control and Configuration Management

Once the product is released an organised method of Change Control and dealing with reported bugs must be created.

- **Request Control**
  - This is a process framework whereby users can request modifications, managers can conduct cost/benefit analysis and developers can prioritise tasks.
- **Change Control**
  - This is a developer's process to recreate a situation reported by a user in order to make appropriate changes to remedy the situation.
- **Release Control**
  - Once changes are made a release control process must be adhered to. Does the change warrant going through system test group etc..

1.8.1 Configuration Management

Within a system the security administrator should formally track versions of code and configurations.

- **Configuration Identification**
  - This is the documentation of the configuration on systems.
- **Configuration Control**
  - If changes to the configuration become necessary they must be made in accordance to a change control process.
- **Configuration Status Accounting**
  - Formalised procedures to track of all authorised changes that take place.
- **Configuration Audit**
  - Periodic audits to ensure the production system is consistent with the accounting records and that unauthorised changes have not occurred.

1.9 Software Testing

1.9.1 White box testing

White box testing uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to identify all paths through the software. The tester chooses test case inputs to exercise paths through the code and determines the appropriate outputs.

While white box testing is applicable at the unit, integration and system levels of the software testing process, it is typically applied to the unit. While it normally tests paths within a unit, it can also test paths between units during integration, and between subsystems during a system level test. Though this method of test design can uncover an overwhelming number of test cases, it might not detect unimplemented parts of the specification or missing requirements, but one can be sure that all paths through the test object are executed.

Typical white box test design techniques include:

- Control flow testing
- Data flow testing
- Branch Testing.

1.9.2 Black box testing

Black box testing takes an external perspective of the test object to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid inputs and determines the correct output. There is no knowledge of the test object's internal structure.

This method of test design is applicable to all levels of software testing: unit, integration, functional testing, system and acceptance. The higher the level, and hence the bigger and more complex the box, the more one is forced to use black box testing to simplify. While this method can uncover unimplemented parts of the specification, one cannot be sure that all existent paths are tested.

1.9.3 Grey Box Testing

Grey box testing involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black box level. Manipulating input data and formatting output do not qualify as grey box, because the input and output are clearly outside of the "black box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey box, as the user would not normally be able to change the data outside of the system under test. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages.

1.10 Security Control Architecture

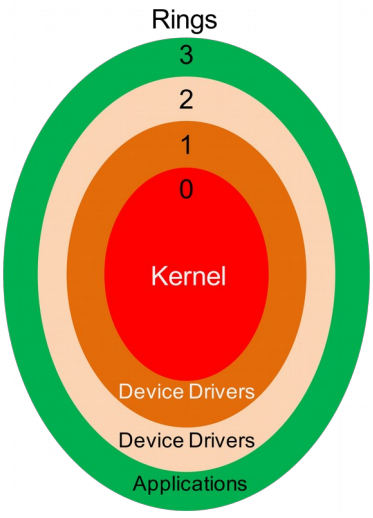
1.10.1 Process Isolation

Process isolation is a set of different hardware and software technologies designed to protect each OS process from other processes. It does so by preventing process A from writing into process B.

Process isolation can be implemented by with virtual address space, where process A's address space is different from process B's address space, preventing A to write into B.

Security is easier to enforce by disallowing interprocess memory access, than compared to less secure architectures (such as DOS) in which any process can write to any memory in any other process.

1.10.2 Protection Rings



Protection rings, are a mechanism to protect data and functionality from faults (fault tolerance) and malicious behaviour (computer security). This approach is diametrically opposite to that of capability based security.

Computer Operating Systems (OS) provide different levels of access to resources. A protection ring is one of two or more hierarchical levels or layers of privilege within the architecture of a computer system. This is generally hardware enforced by some CPU architectures that provide different CPU modes at the firmware level. Rings are arranged in a hierarchy from most privileged (most trusted, usually numbered zero) to least privileged (least trusted, usually with the highest ring number). On most OSs, Ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and memory.

Special gates between rings are provided to allow an outer ring to access an inner ring's resources in a predefined manner, as opposed to allowing arbitrary usage. Correctly gating access between rings can improve security by preventing programs from one ring or privilege level from misusing resources intended for programs in another. For example, spyware running as a user program in Ring 3 should be prevented from turning on a web camera without informing the user, since hardware access should be a Ring 1 function reserved for device drivers. Programs such as web browsers running in higher numbered rings must request access to the network, a resource restricted to a lower numbered ring.

1.11 Service Level Agreement (SLA)

An SLA is a negotiated agreement between two parties where one is the customer and the other is the service provider. This can be a legally binding formal or informal "contract".

The SLA records a common understanding about services, priorities, responsibilities, guarantees, and warranties. Each area of service scope should have the "level of service" defined. The SLA may specify the levels of availability, serviceability, performance, operation, or other attributes of the service, such as billing. The "level of service" can also be specified as "target" and "minimum," which allows customers to be informed what to expect (the minimum), whilst providing a measurable (average) target value that shows the level of organisation performance. In some contracts, penalties may be agreed upon in the case of non-compliance of the SLA (but see "internal" customers below). It is important to note that the "agreement" relates to the services the customer receives, and not how the service provider delivers that service.

2. Open Web Application Security Project (OWASP)

OWASP is an open community dedicated to enabling organistions to conceive, develop, acquire, operate, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security.

2.1 OWASP Top-10 Web Vulnerability List (2017)

OWASP produce a Top-10 vulnerability list every two years or so. Here is the current list. There is a current Top 10 - 2016 Data Call.

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Broken Access Control
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Insufficient Attack Protection
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Under-protected APIs.

Reference

OWASP. 2016. OWASP Top 10 Release Candidate 1. 2017. [online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2017_Release_Candidate_1 [accessed: 20 Sept 2017].

2.2 OWASP Top-10 Mobile Vulnerability List (2016)

OWASP have also represented the mobile application threat landscape from an industry poll of vulnerability statistics.

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorisation
- M7: Poor Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality.

Reference

OWASP. 2016. Mobile Top 10 2016-Top 10 [online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10 [accessed: 20 Sept 2017].

2.3 OWASP Pro Active Controls 2016

OWASP have produces a set of Top 10 Proactive Controls to raise awareness about application security by describing the most important areas of concern that software developers must be aware of. OWASP encourage the use of the controls to aid in the building of secure software.

- C1: Verify for Security Early and Often
- C2: Parameterise Queries
- C3: Encode Data
- C4: Validate All Inputs
- C5: Implement Identity and Authentication Controls
- C6: Implement Appropriate Access Controls
- C7: Protect Data
- C8: Implement Logging and Intrusion Detection
- C9: Leverage Security Frameworks and Libraries
- C10: Error and Exception Handling.

2.3.1 C1: Verify for Security Early and Often

Across many organisations security testing is performed outside the development testing loops, following a '***scan – then - fix***' approach. The security team runs a scanning tool or conducts a pen test, triages the results, and then presents the development team a list of vulnerabilities to be fixed, referred to as '*the hamster wheel of pain*'.

A better way is to make security testing an integral part of a developer's software engineering practice. Security testing should not be left to the end of a project, it needs to be verified early and often, whether through manual testing or automated tests and scans as the software is being developed.

2.3.2 C2: Parameterise Queries

SQL Injection is one of the most dangerous web application risks. SQL Injection is easy to exploit with many open source automated attack tools available. SQL injection can be simply devastating to an application by the simple insertion of malicious SQL code into a web application meaning that the entire database could potentially be stolen, dropped, or modified or even used to run dangerous OS commands against the system host.

To mitigate SQL injection, untrusted input should be prevented from being interpreted as part of a SQL command. The best way to do this is with the programming technique known as '*Query Parameterisation*'. In this case, the SQL statements are sent to and parsed by the database server separately from any parameters.

2.3.3 C3: Encode Data

Encoding involves translating special characters into some equivalent form that is no longer dangerous in the target interpreter. Encoding is needed to stop various forms of injection including command injection (Unix command encoding, Windows command encoding), LDAP injection (LDAP encoding) and XML injection (XML encoding).

2.3.4 C4: Validate All Inputs

Any data which is directly entered by, or influenced by, users should be treated as untrusted. An application should check that this data is both syntactically and semantically valid before using it in any way. Additionally, the most secure applications treat all variables as untrusted and provide security controls regardless of the source of that data. Syntax validity means that the data is in the form that is expected.

2.3.5 C5: Implement Identity and Authentication Controls

Authentication is the process of verifying that an individual or an entity is who it claims to be and is commonly performed by submitting a username or ID and one or more items of private information that only a given user should know.

Session Management is a process by which a server maintains the state of an entity interacting with it. This is required for a server to remember how to react to subsequent requests throughout a transaction. Sessions are maintained on the server by a session identifier which can be passed back and forth between the client and server when transmitting and receiving requests. Sessions should be unique per user and computationally impossible to predict.

Identity Management is a broader topic that not only includes authentication and session management, but also covers advanced topics like identity federation, single sign on, password management tools, delegation, identity repositories and more.

2.3.6 C6: Implement Access Controls

Access Control or Authorisation is the process where requests to access a particular feature or resource should be granted or denied. It should be noted that authorisation is not equivalent to authentication. Access control design requirements should be considered at the initial stages of application development as it is difficult and time consuming to re-engineer access control at a later stage of development.

2.3.7 C7: Protect Data

Encrypting data in Transit. When transmitting sensitive data, at any tier of an application or network architecture then encryption of the data in transit must be considered. Transport Layer Security (TLS) is by far the most common and widely supported model used by web applications for encryption in transit.

Encrypting data at Rest. Cryptographic storage is difficult to build securely. It's critical to classify data and determine that data needs to be encrypted, such as the need to encrypt credit cards per the Payment Card Industry (PCI) - Data Security Standard (DSS) compliance standard. Instead of building from scratch consider using open libraries like:

- Google KeyCzar project
- Bouncy Castle
- functions included in SDKs.

A common weakness in encrypting data at rest is using an inadequate key, or storing the key along with the encrypted data. Keys should be treated as secrets and only exist on the device in a transient state, entered by the user so that the data can be decrypted, and then erased from memory. An alternative is to use of specialised crypto hardware such as a Hardware Security Module (HSM) for key management and cryptographic process isolation.

2.3.8 C8: Implement Logging and Intrusion Detection

Application logging should not be an afterthought or limited to debugging and troubleshooting. Logging is also used in other important activities:

- Application monitoring
- Business analytics and insight
- Activity auditing and compliance monitoring
- System intrusion detection
- Forensics.

Logging and tracking security events and metrics helps to enable '*attack driven defence*': Security testing and controls must be aligned with real world attacks against systems. For example a PCI DSS audit log will contain a chronological record of activities to provide an independently verifiable trail that permits reconstruction, review and examination to determine the original sequence of attributable transactions. It is important not to log too much, or too little. Make sure to always log the timestamp and identifying information like the source Internet Protocol (IP) address and user-ID, but be careful not to log private or confidential data. Protect logs from '*log Injection*' by making sure to perform encoding on untrusted data before logging it.

2.3.9 C9: Leverage Security Frameworks and Libraries

It is a waste of time '*re-inventing the wheel*' when it comes to developing security controls and can lead to massive security holes. Secure coding libraries and software frameworks with embedded security help software developers guard against security related design and implementation flaws. It is preferable to have developers take advantage of what they're already using instead of forcing yet another library on them. Web application security frameworks to consider include:

- Spring Security
- Apache Shiro
- Django Security
- Flask security.

2.3.10 C10: Error and Exception Handling

Implementing correct error and exception handling is an important part of defensive coding, critical to making a system reliable as well as secure. Mistakes in error handling can lead to different kinds of security vulnerabilities.

- It is recommended to manage exceptions in a centralised manner to avoid duplicated try/catch blocks in the code, and to ensure that all unexpected behaviours are correctly handled within the application.
- Ensure that error messages displayed to users do not leak critical data, but are still verbose enough to explain the issue to the user.
- Ensure that exceptions are logged in a way that gives enough information for Q/A, forensics or incident response teams to understand the problem.

OWASP. 2016. OWASP Top 10 Proactive Controls 2016 [online]. Available: https://www.owasp.org/index.php/OWASP_Proactive_Controls [accessed: 20 Sept 2017].

2.4 Controls mapping to the Top-10 2016

The table below maps the address vulnerabilities to the implementable controls to address them.

Control	Vulnerability addressed
C1: Verify for Security Early and Often	All Top 10
C2: Parameterise Queries	Injection Weak Server side controls
C3: Encode Data	Injection Cross Site Scripting (XSS) Client Side Injection
C4: Validate All Inputs	Injection (in part) Cross-Site_Scripting_(XSS) (in part) Unvalidated Redirects and Forwards Security Decisions Via Untrusted Inputs (in part)
C5: Identity and Authentication Controls	Broken Authentication and Session Management Poor Authorisation and Authentication
C6: Implement Access Controls	Insecure Direct Object References Missing Function Level Access Control Poor Authorisation and Authentication
C7: Protect Data	Sensitive Data Exposure Insecure Data Storage
C8: Implement Logging and Intrusion Detection	All Top 10 Unintended Data Leakage
C9: Leverage Security Features and Libraries	All Top 10
C10: Error and Exception Handling	All Top 10

Reference

OWASP. 2016. OWASP Proactive Controls mapping [online]. Available: https://www.owasp.org/index.php/OWASP_Proactive_Controls?refresh=123#tab=Top_10_Mapping_2016 [accessed: 20 Sept 2017].