Unit 1: Introduction to Secure Software Development

What we did in this unit:

* We looked in depth at the different project management methodologies in the context of developing secure software
* We learned about the different types of UML diagrams
* We discovered what it means to possess risk awareness

**Introduction to Secure Software Development Lecturecast**

**Cybercrime**

**How Big of a problem is Cybercrime?**

Preventing unauthorised access to systems has become increasingly difficult with the technological advancements that are being made on a daily basis.

Hackers are constantly coming up with innovative solutions to exploit any potential vulnerabilities in a system.

Companies must not only be alert to external threats they have to also expect threats from people inside of the company.

Cybercrime is an exponentially growing problem.

Accenture a leading IT company have stated that staying ahead of attackers is a constant battle and the cost of doing so is unsustainable.

In a world where companies have to be self-sustainable the cost of preventing attackers from gaining unauthorised access can lead companies to go bust.

Because the cost and legal implications for a company it is important that any software produced by a company is as secure as possible.

The Information Commissioner’s Office is responsible for ensuring the rights of the public in relation to their interactions with online services

The GDPR outlines the data protection principles, rights and rules that an organisation must follow (there are legal ramifications for not following the GDPR for example if the company has been found liable for a data breach that resulted in your personal data being leaked when they shouldn’t have kept this data then they will receive a significant fine)

**What is a Standard, in General?**

* A standard is a set of best practices outlined by a governing body
* In the context of IT security, a standard is a series of good practices that it is strongly recommended to follow
  + These practices help to ensure the security of a system
* Standards are developed by independent parties who don’t possess a bias.
  + Ranging from organisations to governments

**Standardisation bodies for software development**

* W3C
  + The W3C standards support application development
  + The World Wide Web Consortium contribute to an open web platform of recommendations
  + Founded in 1994
  + Led by Time Berners-Lee the inventor of the world wide web
* ISOC
  + The Internet Society contributes to standards on how the internet can be developed and used
  + Founded by Vint Cerf and Robert Khan
  + Created the Network Time Security Protocol which ensures all time on the internet is synced, this is relevant to the Kerberos protocol as it uses time for authenticating connection to a server
* NIST
  + The National Institute of Standards and Technology is responsible for contributing standards which relate to technology.
  + Founded in 1901
* ISO/IEC
  + The International Organization for Standardisation (ISO) and the International Electrotechnical Commission (ICE) contribute to standards on information technology and to information and communications technology
* IETF
  + The Internet Engineering Task Force is responsible for contributing protocols which govern how network systems operate and communicate.
* IEEE
  + The Institute of Electrical and Electronics Engineers research network operations and practises
* OMG
  + The Object Management Group contribute to the creation of new technology standards

**Industry Practises**

**The Open Web Application Security Project (OWASP)**

* Some notable contributors to OWASP are Panasonic, Accenture and Allstate
* OWASP provides you with a checklist of best practices to improve software security
  + I am familiar with OWASP as we have to do annual training learning about OWASP
  + Where I work, we have automation that raises OWASP bugs in Jira that get categorised and assigned to the appropriate team

**The OWASP Top 10 Proactive Controls**

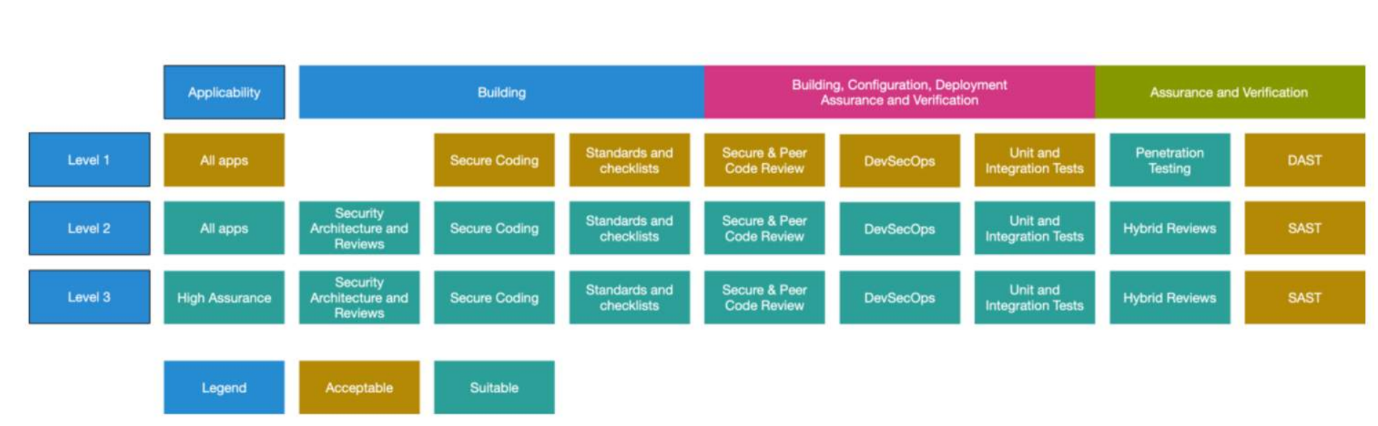
1. C1: Define security requirements
2. C2: Leverage security frameworks and libraries
3. C3: Secure database access
   * + Secure Queries
     + Secure Configuration
     + Secure Authentication
     + Secure Communication
4. C4: Encode and escape data
   * + Encode 🡪Encrypting data to make it difficult to decode
     + Escaping data 🡪 Adding a character before a character in a string so that it isn’t misinterpreted (E.G. a backslash before double quotes)
     + Used to prevent injection attacks
5. C5: Validate all inputs
6. C6: Implement digital identity
7. C7: Enforce access controls
8. C8: Protect data everywhere
9. C9: Implement security logging and monitoring
10. C10: Handle all errors and exceptions

**Weakness identified by OWASP**

* Insufficient logging
* Insufficient monitoring of the systems state
  + Because of this a denial-of-service attack may not be noticed until its too late
* Broken access control
  + Allow users to have full unlimited access when they should only have limited access
* Cross-Site Scripting
  + Allows an attacker to impersonate a user of the system to run malicious code
* Insecure deserialization
  + Can lead to remote code execution allowing harmful data to be inserted into a system

**The OWASP Application Security Verification Standard (ASVS) Project**

* The latest version of The OWASP ASVS Project was released in March 2019
* Provides a list of requirements for secure software development
* Security verification levels:
  + Level 1
    - The lowest level of security
    - Penetration testable
  + Level 2
    - The recommended level of security for the majority of applications
  + Level 3
    - For applications with highly sensitive data that require the highest level of trust



**Integrating Security**

**Agile Approach**

* An iterative approach to development through sprints
* Allows you to easily adapt to changes
* Developers can return to earlier stages in the life cycle
* Relies on good communication
* At Verint we work in two-week sprints using a kanban board. We have a daily standup meeting as well as sprint planning and sprint retrospective meetings.
* Ideal for software projects which produce modular microservices, which are spread out for development across a team

**Stages of a sprint**

1. Plan
2. Design
3. Build
4. Test
5. Review

**Types of Agile approaches:**

* Scrum
  + Self-organising teams
  + Daily communication
  + Work in sprints
* Paired Programming
  + Two developers work together one writing code and the other reviewing code
* DSDM
  + Rapid application development to develop the system as soon as possible
  + Constant communication
  + Incremental builds
  + Requires training to do effectively
* TDD
  + Write the test cases before writing any code
  + Produces high quality code

**Agile Manifesto**

* Prioritise Individuals and interactions over processes and tools
* Prioritise working software over comprehensive documentation
* Prioritise customer collaboration over contract negotiation
* Prioritise responding to change over following a plan

**Waterfall Model**

* A sequential approach to software development where each phase depends on the outputs of the previous phase
  + The current stage can’t begin until after the previous stage is completed
* Going back to an earlier phase in the project, can be difficult, costly and time-consuming
* Ideal for small teams where the project is predictable

**Stages of the waterfall model**

1. System requirements
2. Software requirements
3. Analysis
4. Program Design
5. Coding
6. Testing
7. Operations

**Challenges of Integrating Security into Waterfall**

* Considered vulnerable and risky
* Little opportunity to accommodate unexpected events or to revise the initial design
* Excludes the client and end-user
* Delays testing until after the project is done
* Problems can remain unnoticed until the project gets closer to completion.

**Spiral approach**

* The spiral approach is a waterfall style model with security measures
* Iterates through the phases of planning, risk analysis, engineering and evaluation repeatedly until complete
* Risk analysis is where risks are identified and solved

**Challenges of Integrating Security into Agile**

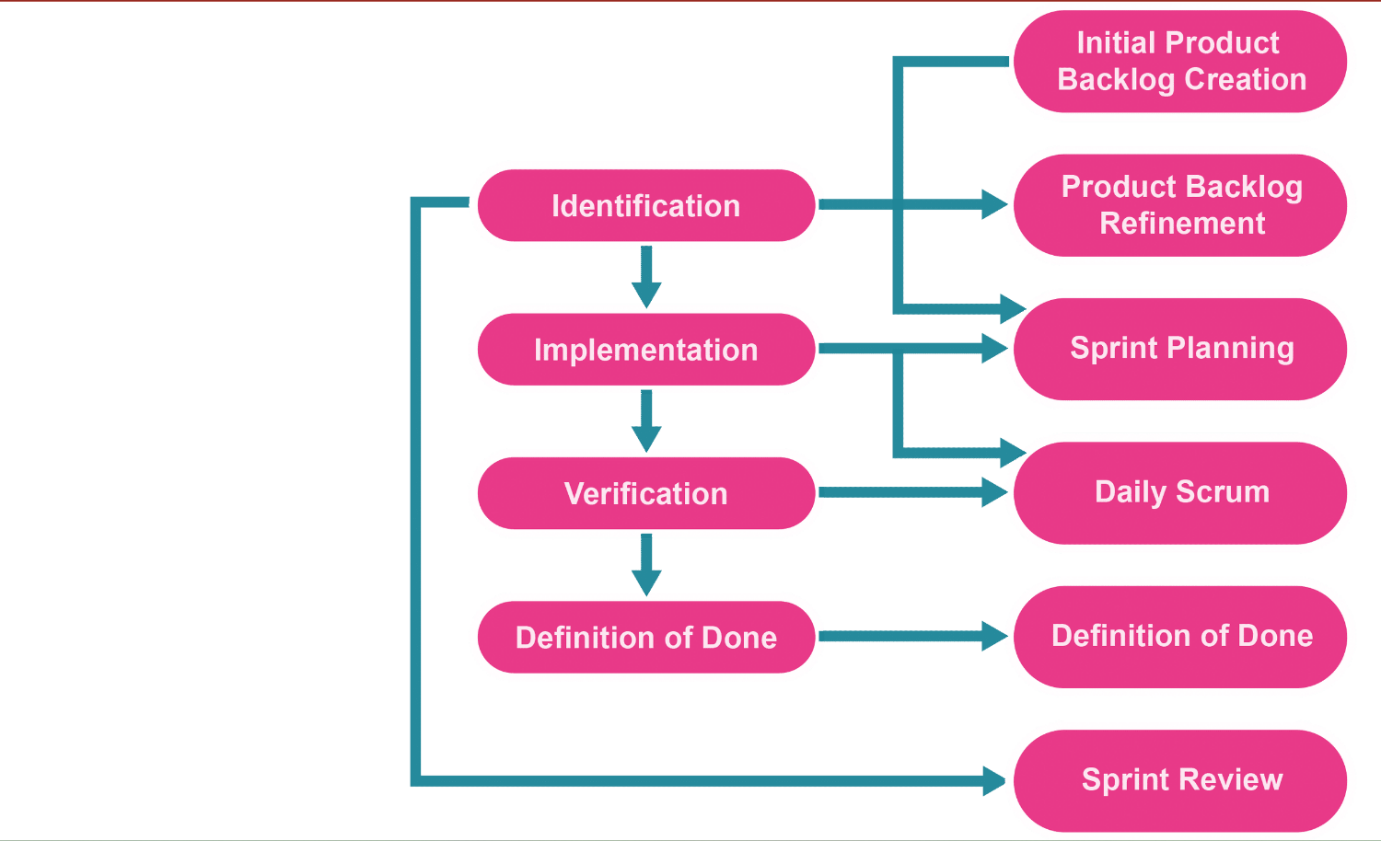
* Short iterations and tight deadlines make it difficult to ensure security is integrated at all stages of development
* May not have time to thoroughly test code
* Software vulnerabilities may not be identified until late into development
* Researchers are constantly looking into way of making Agile more secure

**Steps to integrate security into Agile**

1. Identify important security concerns
2. Identify mechanisms to respond to these concerns
3. Integrate security into each timeboxed sprint
4. Create methods to test each security concern at the end of a sprint
5. Test for security problems at the end of each sprint

**Secure Scrum**

* Secure Scrum applies the stages of Identification, Implementation, Verification and Definition of Done in addition to the traditional Scrum phases.



**SQUARE**

* A 9-step model to integrate security into the DSDM development process
* Expands upon the features of the DSDM phases, with security-specific stages.
* Stage for defining security goals and documenting the security requirements
* Risk assessment performed through the DSDM development process

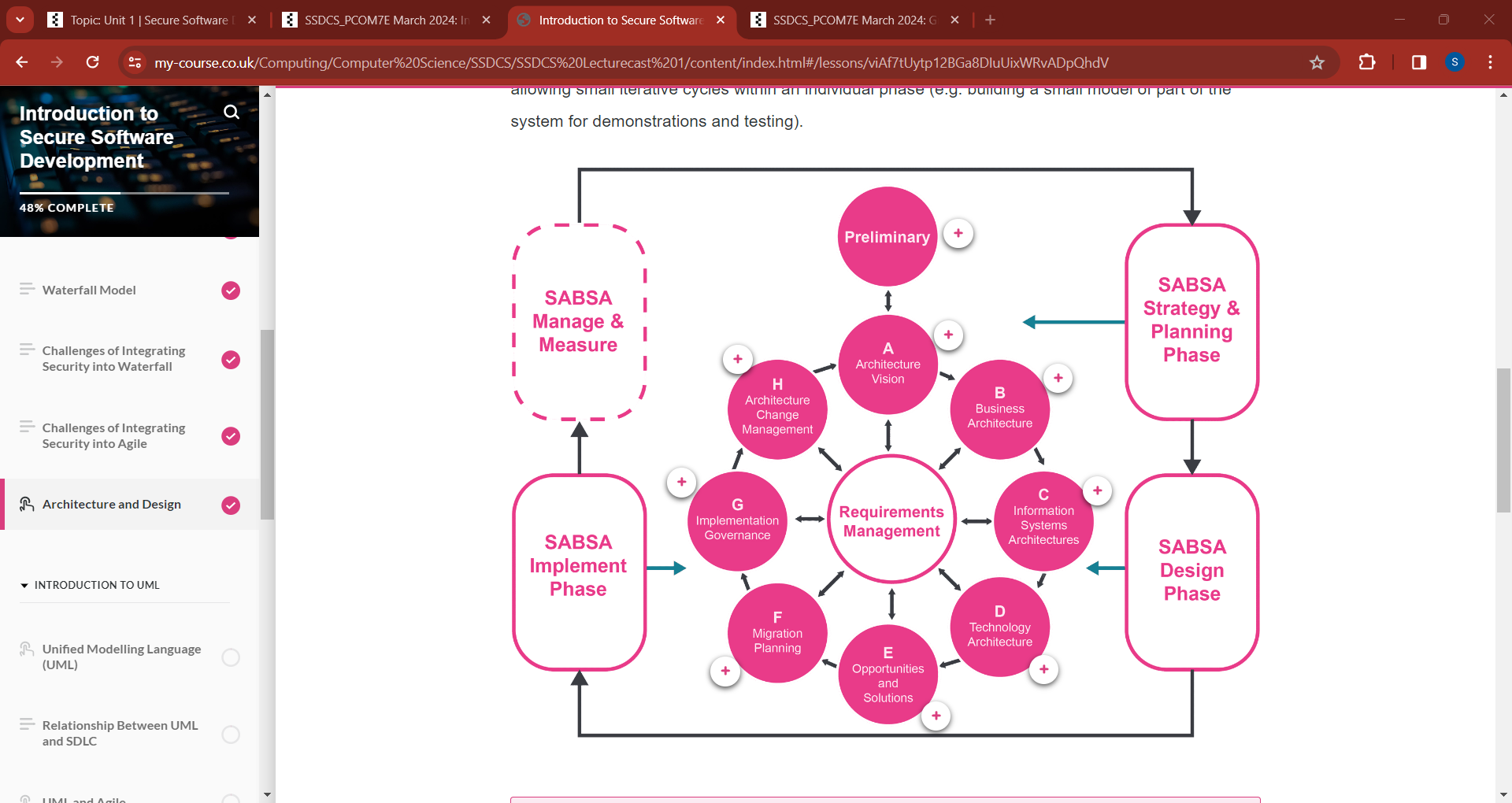
**Architecture and Design**

* It is recommended to design and agree upon the architecture of the prior to starting anything else
* Organisations are often divided in groups based on their functionality
* The open group developed the TOGAF framework which contains standards related to open-source software, architecture certification and system standards
* TOGAF is a tool used by organisations to create their own customised architecture framework
* TOGAF places a lack of emphasis on security
  + Addressed in a white paper by TOGA and SABSA
* Architecture and the SLDC are intrinsically linked
* Design patterns exist in architecture to provide templates or blueprints that help provide solutions or at least starting points to solutions for a problem
* An architectural principle is design reuse (don't reinvent the wheel)
* The architecture will help you decide what tools and technologies to use

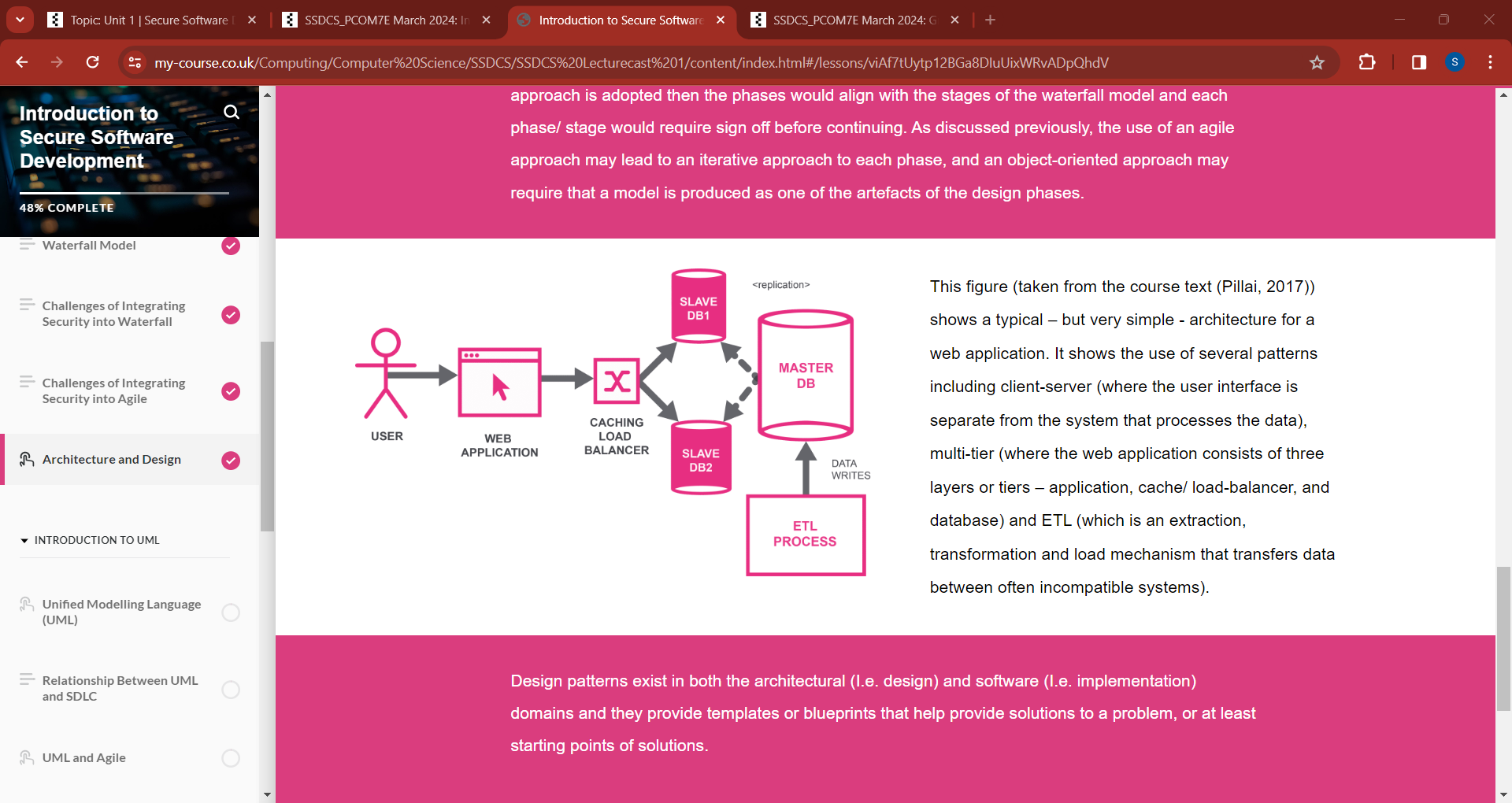
**Architecture Development Method**

1. Preliminary phase
   * Identify stakeholders
2. Architecture vision
   * Position new solution or architecture within the business or organisation structure
   * Create high level proposal to be agreed and approved by stakeholders
   * Produce a project roadmap
3. Business Architecture
   * Business architect or analyst works with stakeholders and users to gather and refine requirements
   * Concentrates on changes to processes, organisations and locations caused by the new project
4. Information Systems Architectures
   * Looks at the information systems aspect of the project
   * Focuses on the data and applications affected by the design
   * Driven by solution and data architects and analysts
5. Technology Architecture
   * Concerns with the technology architecture
   * Driven by technical or infrastructure architects and analysts
   * Sometimes called POLDAT
   * May be iterative
6. Opportunities and Solutions
   * First working models
   * Identify opportunities to implement managed change via a series of transition architectures
   * Create implementation plan
7. Migration Planning
   * How do we deliver or deploy the new systems in the least disruptive way
   * How do we transition users on the new system
   * What training will we provide
   * How do we migrate existing data
8. Implementation Governance
   * The project manager plans and schedules the project
   * The lead architect ensures the implementations comply with the designs and latest standards
9. Architecture Change Management
   * Most import phase of the project
   * Determines success of the overall project
   * Change board consisting of senior stakeholders, the project manager and lead architect review issues and challenges faced

**TOGAF-SABSA**



**Architecture for a simple web application**



**Introduction to UML**

* UML (Unified Modelling Language) is used to represent systems throughout their development
* UML is a graphical language used for system design
* Types of UML diagrams:
  + Structure
    - Static representations of the structure of a system
    - Includes objects, attributes, operations and relationships
  + Behavioural
    - Representations of the dynamic events and activities of the system
    - Capture interactions and collaborations between objects in addition to internal object state
* Examples of UML diagrams:
  + Use case diagram
  + Sequence diagram
  + Class diagram
  + Activity diagram
  + Flow chart
* A UML model is made up of classifiers events and behaviours
  + Classifiers are the set of objects in the system
  + Events are the occurrences that take place in the system
  + Behaviours are reactions to events
* Support communication between project stakeholders
* Support the system design
* Support the maintainability of the system after development
* UML diagrams can be used as a form of documentation

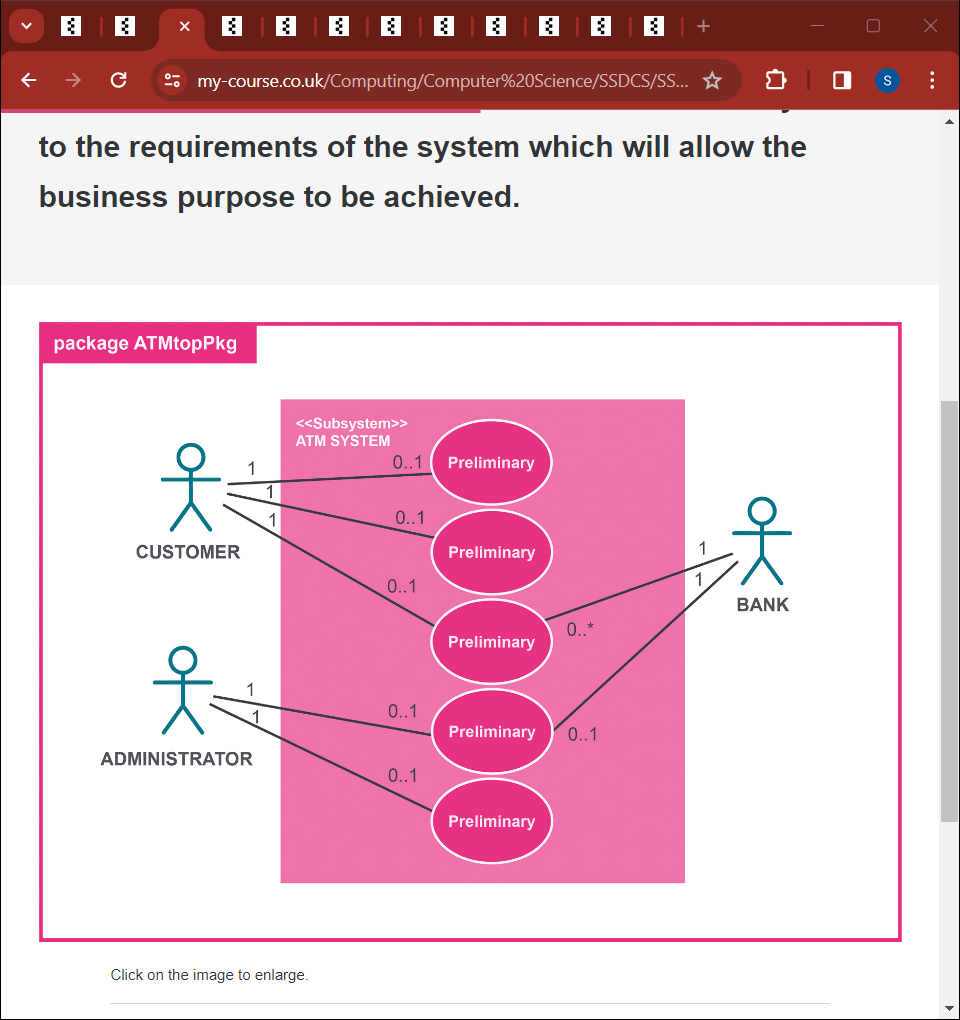
**Relating UML Models to the SDLC**

|  |  |
| --- | --- |
| SDLC Stage | UML Model |
| Requirements | Use Case |
| Design | Class Diagram, Object Diagram, Activity Diagram & Sequence Diagram |
| Development | Activity Diagram & Sequence Diagram |
| Testing | Composite structure diagram & State machine diagram |
| Deployment | Deployment diagram |

**Use Case Diagram**

* Used to show actors interact with a system
* Captures actors, use cases and subjects
* Graphical representation for user stories

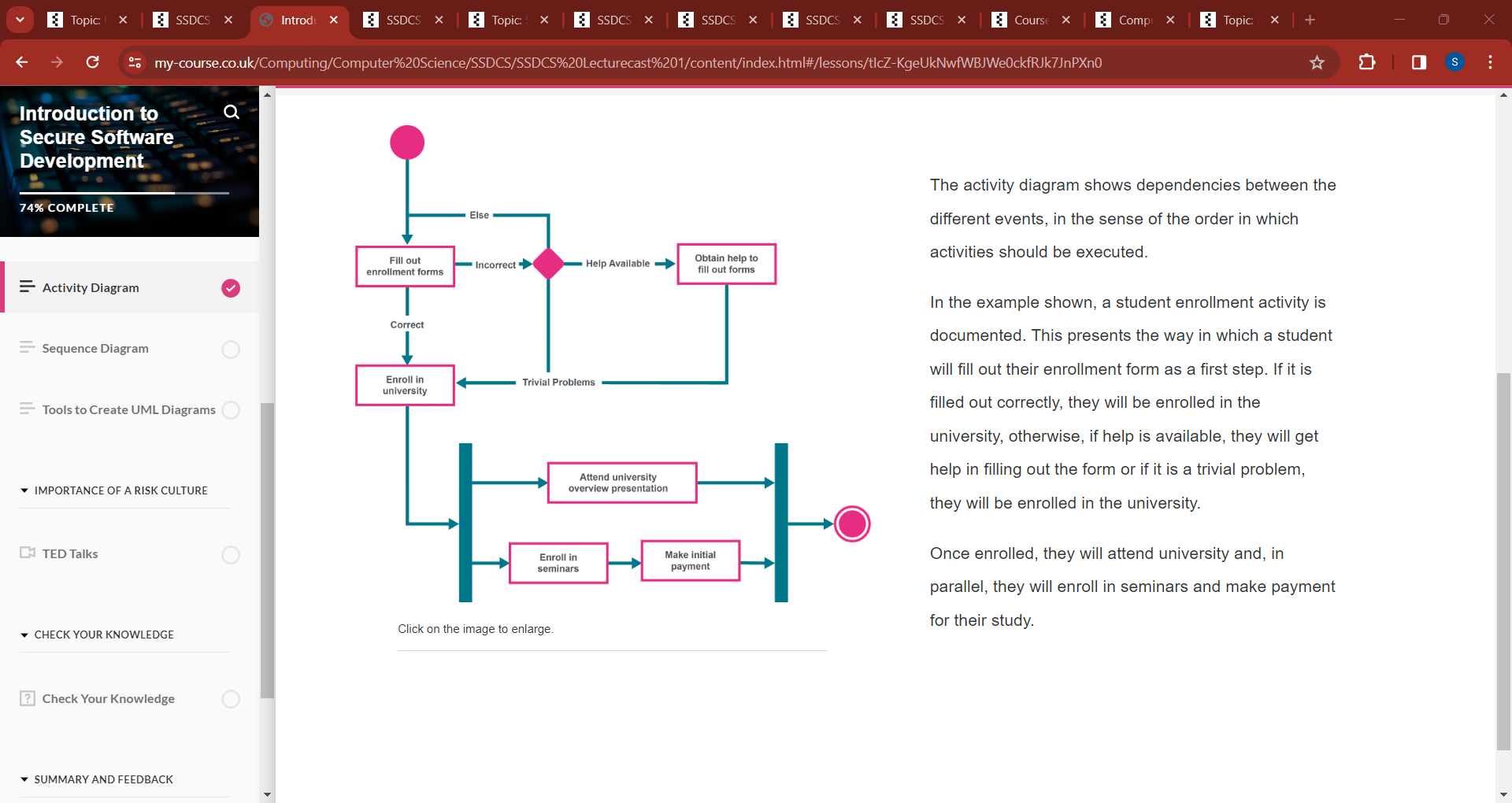
Example Use Case Diagram:



**Activity Diagram**

* Show events that are executed within a system over time
* Show dependencies between events
* Show order activities should be executed

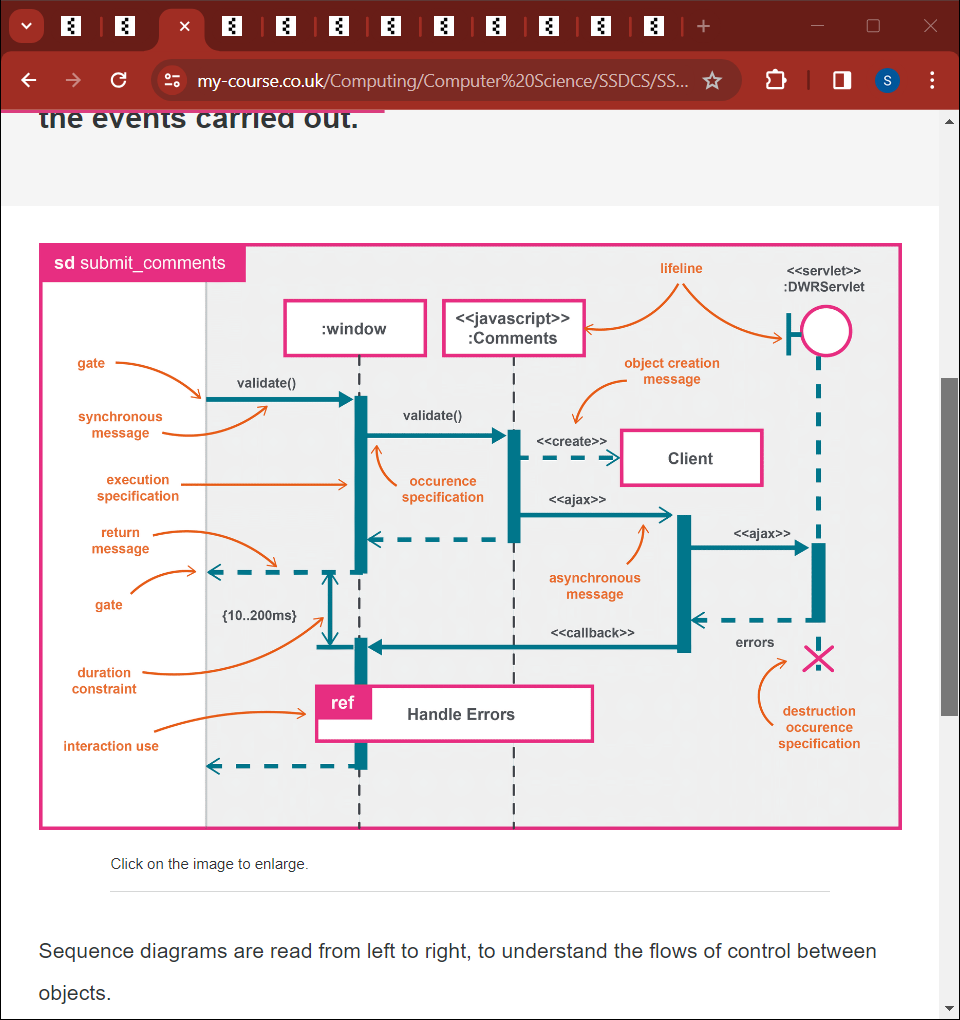
Example Activity Diagram



**Sequence Diagram**

* Shows how events are executed over time
* Shows how messages are distributed between objects in response to an event
* Read from left to right from top to bottom
* Show the flow of an event

Example Sequence Diagram



**Tools to create UML diagrams**

* Draw.io
* Visual Paradigm
* Umberllo
* Lucid chart

**TED Talks**

**Governments don't understand cyber warfare. We need hackers**

* Spoke about how social media is used to recruit terrorists
* Mentioned how governments are slow to react to digital threats
* Spoke about how Anonymous threatened to take down a cartel before backing off due to the threatened body count
* Spoke about peer-to-peer security
* Spoke about educating and recruiting the next generation of hackers
* Spoke about the negatives of becoming a surveillance state

**Hire the hackers!**

* Spoke about the hacking collective anonymous
* Made predictions about the importance of the internet in the near future
* Spoke about the issues involved in cyber security
* Spoke about the hackers profiling project (UN project)
* Spoke about cyber criminals
* Spoke about CarderPlanet (Supermarket for Cybercriminals – similar to the dark web)
* Explained how hackers are vulnerable to exploitation
* Spoke about how hackers should be recruited not incarcerated

**Insider threats to organisations**

* Turncloak 🡪 Someone who intentionally harms the system
* Pawn 🡪 Someone who inadvertently harms the system

**Check your knowledge**

1. What information is represented in a use case diagram?

User stories

1. Fill in the Blank: \_\_\_\_\_\_\_\_\_\_\_\_ can change throughout the SDLC.

Requirements

1. True or False: OWASP is an acronym of Open Web Active Security Project?

False

**Lecture 1 Notes**

* Knowledge from OOP module will help you with this module
* This module will focus on the security aspects of software development (E.G. is access control protected? Is data being transferred securely)
  + AES256 encryption
  + Data encrypted in transit and at rest
* Important to engage in the module as there’s a lot of content to cover
* E-Portfolio is worth 40% of module grade
  + Includes a 1000-word reflection on module
  + All artifacts from the module
    - Things you’ve read, written, output of code
  + Reflection on every unit
  + Strongly suggested to use a reflective model (E.G. Rolfe, Driscoll’s What Model, ERA Cycle, Kolb’s Experimental Learning Cycle, Gibbs Reflective Cycle) as it contributes to structure and presentation mark and academic integrity
  + Gibbs Reflective Cycle suggested
    - Description
      * What is it you’re reflecting on
    - Feelings
      * What are your feelings about it
    - Evaluation
    - Analysis
      * Can you analyse what happened
    - Conclusion
      * What’s your conclusion
    - Action Plan
      * How are you going to move forward in the future based on what you’ve learned
* There is a group assignment in this module
  + Design document
    - 1000-word count
    - Create UML Diagrams
      * Use OMG UML Specification
      * Misuse case diagram
        + How the attacker interacts with the system
    - Make decisions on tools and libraries you may use
    - Security related challenges:
      * Data could be intercepted
      * Someone could gain unauthorised access
      * Data could be deleted
      * Someone could flood the system with uploads
        + Restrict amount of data to be uploaded
      * How are you going to prevent this?
      * Restrict how long somebody has to log into the system
      * Add TFA
      * Limit how long someone can be logged in for require them to be reauthenticated
      * Restricting the number of login attempts
    - Develop an application for a school or online retailer or the international space station
      * Contain role-based access control, encryption, monitoring
      * If shop should be able to add stock, delete stock
      * If school store student records
    - Use inline citations and a reference list
  + Will also have to do a peer review
* Coding assignment is done individually
  + Needs to be developed using Python
  + Data should be stored in data structures in python
  + Application should be run via command line
  + Talk about how you’re meeting the GDPR
    - Can the user delete their own data
  + How are you going to minimise the attack surface?
    - E.G. have less api’s
  + Have authentication
  + Have encryption
    - Fernet library in Python
  + Have monitoring and logging
  + Have ability to turn security on or off
  + Using Linters
  + User should be able to Perform CRUD operations
  + Hacker should be able to simulate brute force attack, denial of service attack and an API injection attack
  + Test system
  + Worth 40% of the module grade
  + Add comments
  + Follow PEP-8 style guide
  + Submit a readme file
* In this module were going to look at how to write secure code and how to design a piece of software
* You will learn how implementing security effects performance
* We will look at the security risks and the types of attacks
  + Man in the middle attack
  + Brute force attack
  + Denial of service attack
* We will learn about the SDLC and learn how to incorporate security into the SDLC
* We will learn about the fundamentals of OOP:
  + Abstraction
  + Inheritance
  + Polymorphism
  + Encapsulation
* Learn about different project management methodologies:
  + Agile
  + Waterfall
  + Spiral
* First six weeks of module will require you to work as part of a team
* Security Risks to a system
  + Who is accessing the system?
  + How are they accessing the system?
  + Where is data stored?
  + How long is data being stored?
  + Is it being deleted?
* Management and logging
  + How do you limit the number of login attempts?
    - Prevent system from being flooded
  + How do you limit the amount of data that can be deleted?
* You will be required to demonstrate your code after submitting the assignment
* Suggested reaching out to teams’ members via email (I sent an email to the members of my team asking if we should create a group chat and reminding them of the deadline for the team contract)
* Team contract has to be done by Monday and emailed to the lecturer
* Introductory Seminar preparation:
  + Collaborative discussion one
    - Select a UML tool
    - Select one of the coding weaknesses identified by OWASP
    - Create a flowchart of the steps that may have led to the weakness occurring
      * Why did this problem happen within our code
    - Week 1 make your post
    - Week 2 comment on your peers
    - Week 3 write your summary post
  + Share E-Portfolio
* OWASP is a body which promotes secure code development
  + Its not a standardisation they only make recommendations
* The OWASP framework supports you in writing secure code
* They have labs to develop skills and a chat bot for support
* OWASP Top 10 Proactive Controls 2021: (https://owasp.org/Top10/)
  + Broken Access Control
  + Cryptographic Failures
  + Injection
  + Insecure Design
  + Security Misconfiguration
  + Vulnerable and Outdated Components
  + Identification and Authentication Failures
  + Software and Data Integrity Failures
  + Security Logging and Monitoring Failures
  + Server-Side Request Forgery
* When designing your code say how you're going to follow the OWASP top 10 proactive controls
  + Shows greater academic integrity and engagement with this module
* All the security capabilities that you enable in your code should come from the OWASP top 10 proactive controls
* Weakness identified by OWASP:
  + Broken Authentication
  + Sensitive Data Exposure
  + XML External Entities
  + Broken Access Control
  + Security Misconfiguration
  + Cross-site Scripting
  + Insecure Deserialization
  + Using Components with Known Vulnerabilities
    - E.G. At Verint we were using mRemoteng to RDP/SSH onto machines but based on intelligence, Verint Security Operations identified this as a high security risk so blocked the application and insisted on using Microsoft’s Remote Desktop Manager
  + Insufficient Logging and Monitoring
* Preparation for next week’s lecture:
  + Create a 2-column table
    - First column
      * The software development stages for Scrum
    - Second Column
      * The processes or recommendations you would make to ensure secure software is developed in each stage
      * E.G. Ensure the right people are allocated to the right role at the beginning of the process (sum master, project manager etc)
      * When requirements change the security reflects the changes
      * Go through each of the stages and think about how to create a secure process
  + Blog post
    - Select five terms from the ISO/IEC 27000 Section 3 terms and definitions
    - Write a 300-word blog post on how people can be managed to overcome cyber security attacks from the inside
      * Training on social engineering
      * Risk assessment on perspective employees
      * How are you protecting vulnerable people?

**Unit 1 Reading**

**Software Architecture with Python**

**Chapter 1 Principles of Software Architecture**

**What is software architecture?**

* Software architecture is a description of the subsystems of components of a software system and the relationships between them

**Software architecture versus design**

* Architecture covers the higher-level structures and interactions in a system
  + The design of the entire system
* Design is about the organisation of parts of the system their subsystems
  + The design at implementation level
  + Which modules to split code into?
  + What classes to assign the different functionality to?
  + Which design patterns to use?
  + How do objects interact with each other at runtime?

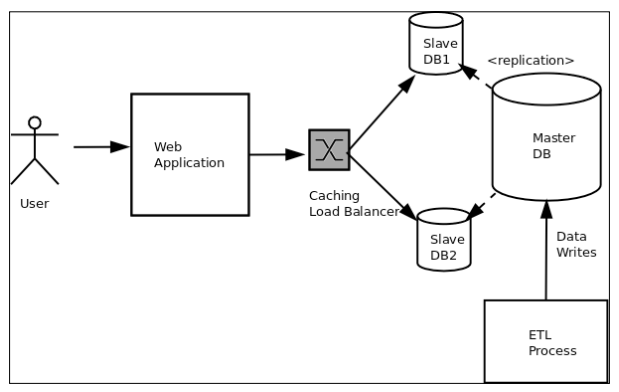
**Aspects of software architecture**

* System
  + A collection of software components to achieve some functionality
* Structure
  + How a set of elements are grouped
* Environment
  + The context in which the system is built
* Stakeholder
  + A person who has a vested interest in the system

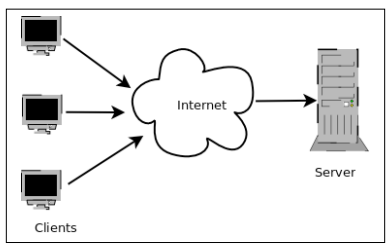
**Characteristics of software architecture**

* An architecture defines the structure of a system
  + Shown through UML diagrams
* An architecture picks a core set of elements required for the core functionality of the system
* An architecture captures early design decisions
* An architecture manages stakeholder requirements
* An architecture influences the organisational structure
* Architecture is influenced by its environment
* An architecture documents the system
* An architecture often conforms to a pattern
  + Client-server
  + Pipes and filters
  + Data-based architectures

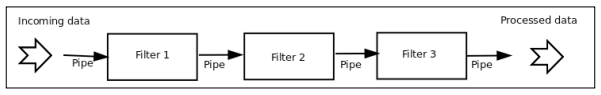
**Example architecture diagram**



**Example of client-server architecture**



**Example of pipe and filters architecture**



**Types of structures**

* Runtime structures
  + Objects created at runtime and how they interact
* Module structures
  + How the code is organised

**Types of architectures**

* Component architecture
* Deployment architecture
* Communication architecture
* Enterprise architecture

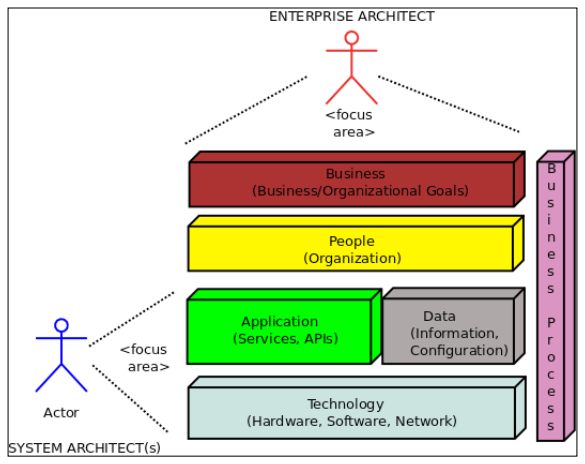
**Importance of software architecture**

* Architecture selects quality attributes for a system
* Architecture enables RAD (rapid application development)
* Architecture allows a system to be built via a series of components
  + Increases reusability
* Architecture helps manage required changes to a system
* Documentation can be shared with stakeholders
* Improves maintainability of system
* Improves reliability of system

**Different types of software architects**

* Technical architect
  + Looks at the core technology used in an organisation
* Security architect
  + Works on the security strategies for an organisation
* Information architect
  + Works to make information available to and from applications to meet the organisations business goals
* Infrastructure architect
* Solution architect
  + Focuses on technology and project scope
* Enterprise architect
  + Looks at the overall business and organisational strategies to guide an organisation
  + Focuses on strategy
* System architect
  + Technology focused
  + Combine systems to create a solution for a specific client

**Schematic diagram for architects**



**What is a quality attribute**

* A measurable and testable property of a system used to evaluate its performance

**Architectural quality attributes**

* Modifiability
  + Extendibility and flexibility of a system
  + Assessed in terms of difficulty, costs and risks associated with changes
* Testability
  + Black box testing
  + Grey box testing
  + White box testing
  + Unit tests
  + Automation tests
  + Integration tests
  + Testing edge cases
  + User testing
  + Regression testing
  + Performance testing
  + Security testing
* Scalability and performance
  + Scaling up and down to meet demand and save costs whilst limiting impact on performance
  + Scale horizontally or vertically
    - Horizontal scaling = scaling by changing the number of nodes running the system
    - Vertical scaling = scaling by changing the number of resources on a single node (E.G Increasing RAM on a VM)
  + Performance measured based on response time, latency and throughput
* Availability
  + Is it able to function when needed
  + Linked to reliability
  + Disaster recovery plan
  + Availability = Mean time between failures / Mean time between failures + Mean time to repair
  + Tied to security
* Security
  + How the system prevents unauthorised access
  + Authentication
    - Singing
    - TFA
  + Authorisation
  + Role based access control
  + CIA (Confidentiality, Integrity and Availability of data)
* Deployability
  + How software can be taken from a development environment to the production environment
  + Nothing to do with the code

**Levels of changes to a system**

* Local
  + Changes only affect a specific element of the system
  + Low cost, low risk changes
* Non-local
  + Changes involve more than one element of the system
  + Medium cost, medium risk changes
* Global
  + Changes involving architecture changes or changes to a significant part of the system
  + High cost, high risk changes

**Relationship between code modifiability, cohesion and coupling**

|  |  |  |
| --- | --- | --- |
| Cohesion | Coupling | Modifiability |
| Low | High | Low |
| Low | Low | Medium |
| High | High | Medium |
| High | Low | High |

**Factors that affect code modifiability**

* Number of lines
* Number of team members working on the code
* Number of external dependencies used
* Wrong use of the module API

**Techniques for availability**

* Fault Detection
  + Detect faults early with monitoring
* Fault recovery
  + How to recover from the fault E.G. rollback
* Fault prevention
  + Anticipating faults and stopping them from happening in the first place

**Factors that determine Deployability**

* Module structure
* Production versus development environment
* Development ecosystem support
* Standardised configuration
* Standardised infrastructure
* Use of containers

**Chapter 6 Security – Writing Secure Code**

* There is an unprecedented amount of data shared across software and hardware systems. Due to the legal and financial ramifications it is vital this data is secure

**Information security architecture**

* Goal is to provide correct levels of access to authorised individuals
* Measured in terms of Confidentiality, Integrity and Availability
  + Confidentiality = only authorised people can access data
  + Integrity = Trustworthiness of data
  + Availability = Can the data be retrieved when necessary

**Security techniques**

* Authentication
  + Verify identity
* Authorisation
  + Verify permission

**Secure coding**

* Secure coding is the practise of developing secure software
* Security should be thought of throughout the SDLC
* Threat modelling should be used to anticipate security threats

**Threat modelling process**

1. Identify important assets
2. Decomposing the application into components
3. Identifying and categorising threats to each asset or component
4. Ranking the threats based on an establish risk model
5. Developing threat mitigation strategies

**Secure Coding process**

1. Definition of area of interest of the application (what parts are critical and need to be secured?)
2. Analysis of software architecture
3. Review of implementation details
4. Verification of logic and syntax
5. Whitebox/Unit testing
6. Blackbox testing

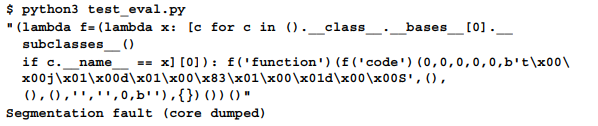
**Common security vulnerabilities**

* Overflow errors
  + Buffer overflow
    - Errors that allow an application to write paste the end or beginning or a buffer.
    - Allow attackers to take control by gaining access to the applications stack or heap memory
  + Integer or arithmetic overflow
    - Occur when a mathematical operation produces a result that is too large for the maximum size of the type used to store it
      * E.G. storing a value that should be a long value as a float
    - Create security vulnerabilities if not handled properly
      * Allows hackers to gain access to heap or stack memory outside the program execution limits
* Unvalidated/Improperly validated input
  + Can be used by hackers to trick a program into accepting malicious code
    - E.G. without prepared statements hacker could delete a database by putting their first name in a form to drop table (this is known as SQL Injection)
  + Add checks to filter and remove malicious code from being inputted to a system
* Improper access control
  + Implement role-based access controls to limit users’ access
  + If not done properly a hacker can impersonate an admin user
* Cryptography issues
  + You should add an audit trail and verification to actions performed by users in a system
  + All data should be encrypted using the latest encryption algorithms
    - Old encryption algorithms should not be used as they have vulnerabilities and can easily be cracked using decryption software
    - AES256
    - DES5
    - SHA
  + Use the latest hashing techniques
    - Add salt to make data harder to decrypt (how blockchain does hashing)
    - SHA instead of MD5
  + Uses HTTPS instead of HTTP
  + Use strong passwords
  + Don’t reuse secrets or keys or hashes
  + Don’t use insecure authentication
  + Keep certificates updated
    - At Verint our signing tool certificate expired recently so we moved to manual signing which was an extensive process
  + Disable password-based access opting for SSH access for specific users only using SSH keys
  + Disable remote root SSH access
* Information leak
  + Reduce the amount of information a hacker can learn about your system due to misconfiguration of a web server
    - Create a custom 404 page and hide certain information such as the name and version of the web servers operating system
    - Protect your directory pages
* Open port access
  + E.G If you should only be able to RDP onto a machine only allow access to inbound connections on port 3389
* Open access to a web server
  + Limit the access to specific IP addresses
  + Set firewall rules and security policies
* Open access to files, folders and databases
* Race conditions
  + When a program has two or more actors attempting to access a resource and the output is order dependent
  + Creates security vulnerability
  + Used in multithreading
    - Requires synchronisation to avoid
* System clock drifts
  + System clock and local clock aren’t synchronised
  + Can cause an error in SSL certificate validation which can be exploited by timing attacks
  + Use NTP to mitigate
* Insecure file/folder operations
  + May not be able to detect someone tampering with the system due to program assumptions
  + Verify a write operation has been successful instead of assuming it worked
  + Assuming local paths are always for local files not symbolic links for system files where the application may not have the required access (resulting in access denied issues)
  + Improperly using sudo in system command
    - Hackers can take advantage of this to gain root access
  + Shared files or folders not done correctly allowing greater access to resources
  + Incorrect permissions for files and folders
    - Should anyone be able to access and write to a file that contains confidential information or should only certain people have view and or write access
  + Using unsafe serialisation and deserialization of code or data

**Is Python secure?**

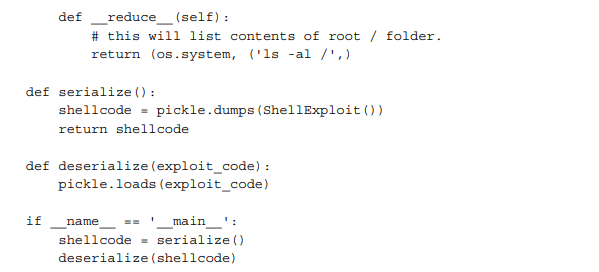
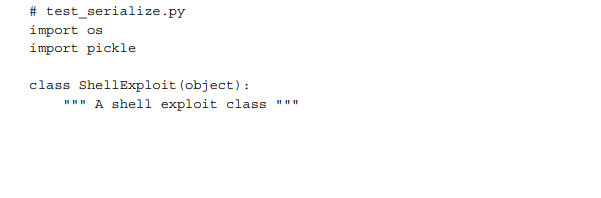
* A lot of the security vulnerabilities from Python 2 have been patched in Python 3
* In Python 2 input data in evaluated as an expression without doing any checks and is printed allowing global variables to be seen by hackers. This exploit has been fixed in Python 3 but lots of company’s legacy applications will be written in Python 2.
* Eval command can be used to execute commands (E.G ls -a)
  + Can be used to run malicious code
  + Can be used to crash the python interpreter and gain control over the system

Python 3 Example to crash the interpreter with eval (this could be used in the module assignment)

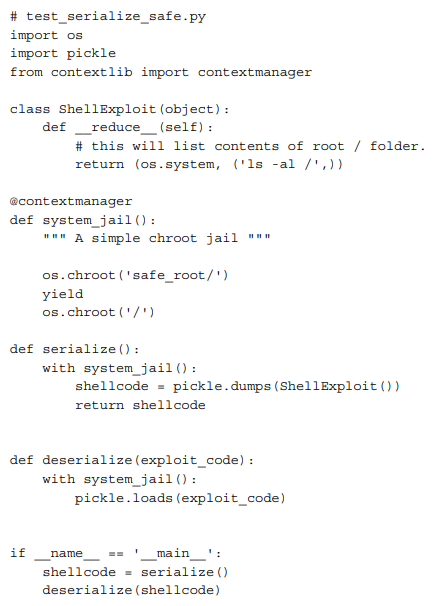


* Overflow errors in Python 2 with range and xrange when converting from int to long
  + Any overflow errors which aren't correctly handled could lead to buffer overflow exceptions which an attacker can take advantage of to gain control over the system

Using the Python pickle module to exploit the system and list all contents of the root folder (Useful for assignment)



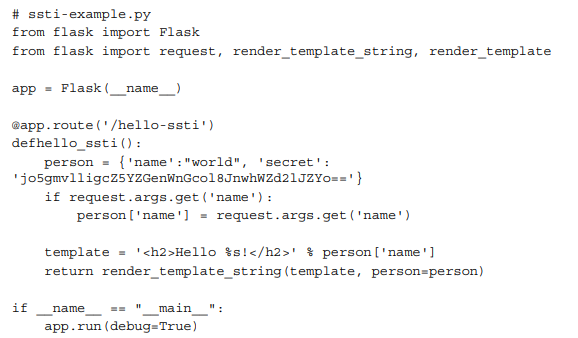
Code to prevent this exploit



**Server-Side Template Injection**

* An attack that uses the server-side templates of common web frameworks as an attack vector
* Used to figure out the internal of a web application, execute shell commands and comprise servers

Flask Example

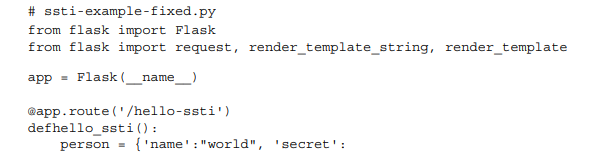
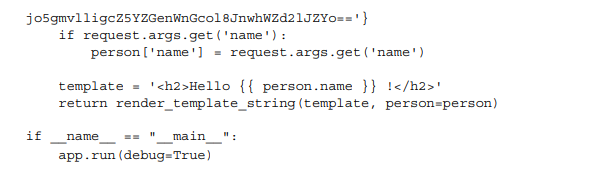


The URL used for the attack is as follows:

http://localhost:5000/hello-ssti?name={% for item in person %}<p>{{

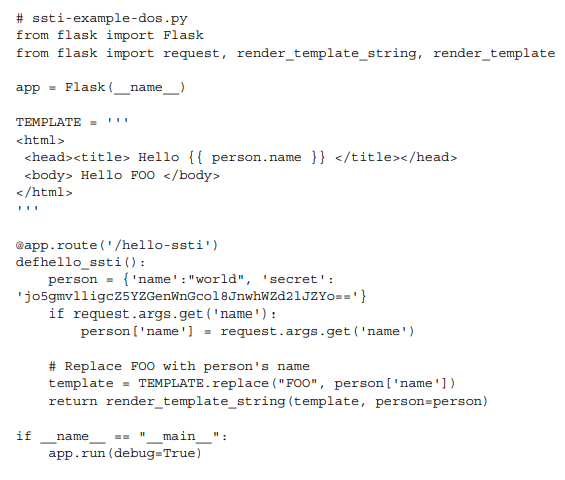
item, person[item] }}</p>{% endfor %}

How to fix the exploit



**Denial of Service**

* Attackers target vulnerable routes in web application sending large packets to crash a server



URL for this attack is http://localhost:5000/hello-ssti?name=Tom

{{ 100\*100000000 }}.

**Cross-Site Scripting**

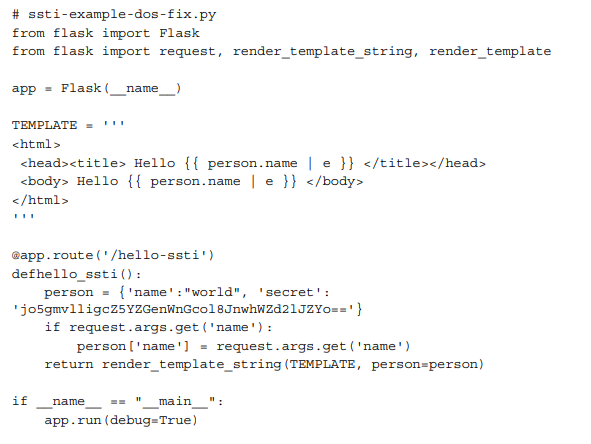
* XSS is where hacker inject malicious scripts in your server’s code to take control of it

The URL used for this attack is as follows:

http://localhost:5000/hello-ssti?name=Tom<script>alert("You are under

attack!")</script>

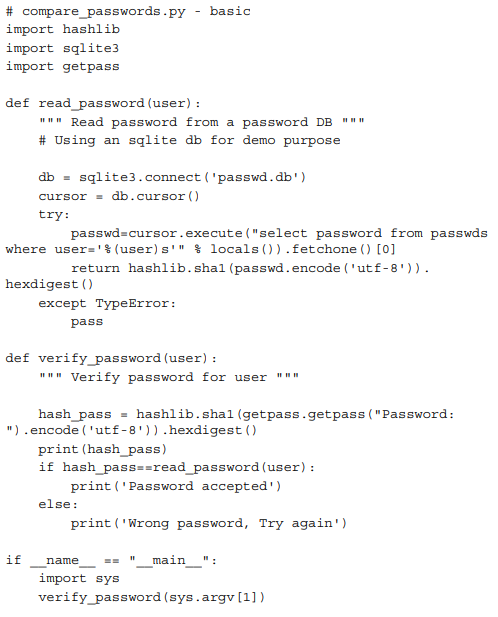
Code to mitigate DOS attacks and XSS attacks



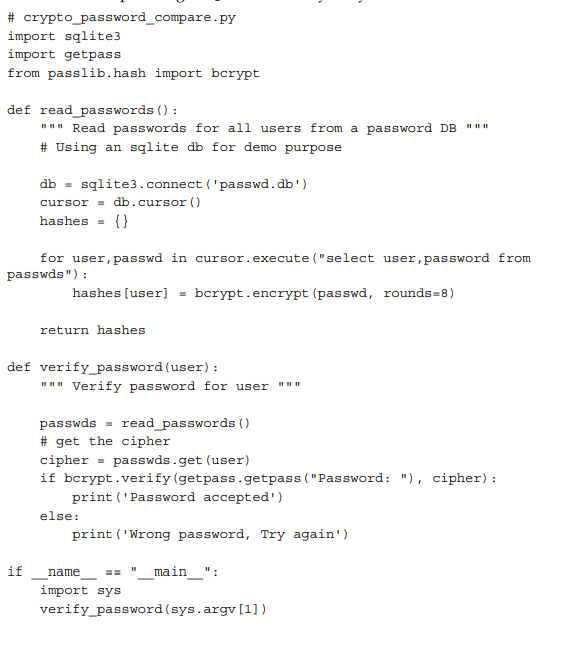
**Strategies for security in Python**

* Use raw input instead of input for reading input
* Do type conversions and validation manually
* Add exception handling
* Don’t use eval or exec
* Don’t use pickle for serialization use JSON or YAML
* Guard again integer overflows
* Use template strings for string formatting
* Use with when operating with files to ensure they are closed
* Validate sensitive information like passwords
* Avoid storing sensitive data local to functions
* Avoid race conditions and thread deadlock
  + Protect resources that can be writable concurrently (threading.Lock)
  + Protect resources that that need to be serialized (threading.BoundedSemaphore))
  + Use condition objects to synchronise threads waiting on a condition or function (threading.Condition)
* Keep your system up to date

Example bad password validation code



Example good password validation code



**Secure coding strategies**

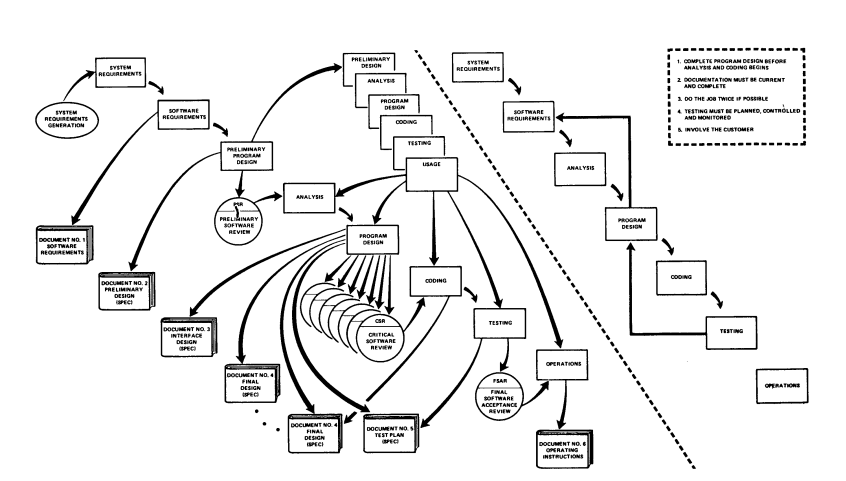
1. Validate Input
2. Keep it simple
3. Principle of least privilege
4. Sanitize data
5. Authorize access
6. Perform effective QA
7. Practise defence in layers
8. Define Security requirements
9. Model threats
10. Architect and design for security policies

**Software Design Patterns Recommendation : A Systematic Literature Review, 2019 International Conference on Frontiers of Information Technology**

* Design patterns aid with reusability, modularisation, quality and consistency of code and design
* Design patterns are grouped into 3 categories:
  + Structural
    - Adapter Pattern
  + Behavioural
    - Observer Pattern
    - Strategy Pattern
    - Visitor Pattern
  + Creational
    - Abstract Factory
    - Template Pattern
    - Singleton
* Stressed the importance of selecting the right design pattern for your use case
* I would question the validity of there results as they only analysed approximately 6% of the sources they found (they exclude 94.05% of results)
* The fact they only used 4 websites for finding papers means their conclusion could be inaccurate
* The limited number of search terms they used would also of had a significant effect on their results
* The fact that a large percentage of their data is from conferences is particularly concerning as they should have collected data from a range of sources.
* There literature review ignores other design patterns such as the build pattern and lightweight pattern

**Managing the Development of Large Software Systems**

* Dr Winston Royce used his experience managing large software projects to advise on how to ensure the success of a project
* As it was written in 1970 his ideas appear rather dated
  + He is talking about the waterfall method which is significantly outdated now
* I don’t agree with his assessment that there are only two essential steps for delivering a small project I would argue regardless of the size of the project the format you follow should be the same (E.G. just because it’s a small project only for internal use doesn’t mean it shouldn’t be treated any differently. What if in the future they decide market the software?)
* It would have been less bias if he got other developers to contribute to the presentation to remove any bias
* While Dr Royce’s diagram would be hugely beneficial to companies back then it could be considered overly complex for people who aren’t as knowledgeable



**Steps involved in program development of large software projects**

1. Design
2. Document the design
3. Do it twice
4. Plan control and monitor testing

**Weaknesses in OWASP Top Ten**

**A1 – Injection**

* Command Injection
* OS Command Injection
* Argument Injection
* SQL Injection
* LDAP Injection
* XML Injection
* Expression Language Injection
* Improper Neutralization of Special Elements in Data Query Logic

**A2 – Broken Authentication**

* Improper Authentication
* Plaintext storage of a password
* SFA
* Session Fixation
* Insufficient protected credentials
* Unprotected transport of credentials
* Insufficient session expiration
* Unverified Password Change
* Weak Password recovery mechanism

**A3 – Sensitive Data Exposure**

* Storage of File With Sensitive Data Under FTP Root
* Improper Certificate Validation
* Missing Encryption of Sensitive Data
* Cleartext Storage of Sensitive Information
* Cleartext Transmission of Sensitive Information
* Key Management Errors
* Missing Cryptographic Step
* Inadequate Encryption Strength
* Use of a Broken or Risky Cryptographic Algorithm
* Use of Weak Hash
* Exposure of Private Personal Information to an Unauthorized Actor

**A4 – XML External Entities**

* Improper Restriction of XML External Entity Reference
* XML Entity Expansion

**A5 – Broken Access Control**

* Path Traversal
* Improper Access Control
* Improper Authorization
* Forced Browsing
* Authorization Bypass Through User-Controlled Key

**A6 – Security Misconfiguration**

* Generation of Error Message Containing Sensitive Information
* Exposure of Information Through Directory Listing

**A7 – Cross-Site Scripting**

* Improper Neutralization of Input During Web Page Generation

**A8 – Insecure Deserialization**

* Deserialization of Untrusted Data

**A9 – Using Components with Known Vulnerabilities**

* Using out of date libraries

**A10 – Insufficient Logging & Monitoring**

* Omission of Security-relevant Information
* Insufficient Logging

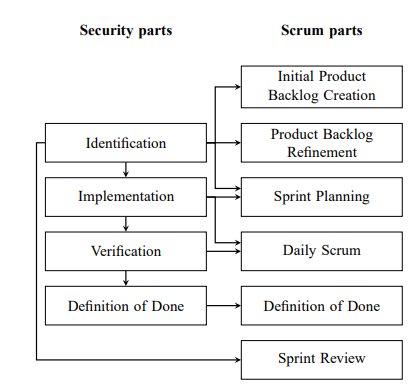
**Secure Scrum: Development of Secure Software with Scrum, in Proc. of the 9th International Conference on Emerging Security Information, Systems and Technologies.**

* Security issues should be considered throughout the SDLC
* Secure Scrum is a variation Scrum that focuses on the development of secure software
* Secure Scrum allows you to spot security issues, implement security features and verify implementations
* The goal of secure scrum is to achieve the appropriate level of security for a software project
  + Software needs to be secured until it is no long profitable for an intruder to find and exploit a vulnerability
* Secure Scrum offers a way identify security relevant parts of a project and judge the attractiveness of attack vectors in the sense of ease of exploitation
* Secure Scrum helps developers identify appropriate security testing means for security relevant parts of a software project.
* Secure Scrum offers a way to include external resources such as security consultants into the project without breaking the characteristics of Scrum

**Components of secure scrum**

* Identification component
  + Identify security issues during software development
  + Security issues so be marked with S-Tags in the product backlog
  + User stories marked according to loss value
* Implementation component
* Verification component
  + Testing the software with a focus on IT security
* Definition of Done component

**Secure Scrum Diagram**



**ISO/IEC Standard 27000 Section 3**

* Access control = Limit access to authorised individuals
* Attack = Attempted authorised access
* Authentication = Verify identity
* Availability = accessible on demand
* Confidentiality = Only authorised people can access information
* Information security = integrity, availability and confidentiality of information is ensured
* Integrity = Accurate & complete information
* Reliability = Intended behaviour and results
* Vulnerability = Weakness that can be exploited

**Collaborative Discussion 1: UML flowchart**

**What OWASP do?**

OWASP are an organisation that provide you with a checklist of best practices on how to improve software security.

**OWASP 2021 Top 10 (OWASP, 2021)**

1. A01: Broken Access Control
2. A02: Cryptographic Failures
3. A03: Injection
4. A04: Insecure Design
5. A05: Security Misconfiguration
6. A06: Vulnerable and Outdated Components
7. A07: Identification and Authentication Failures
8. A08: Software and Data Integrity Failures
9. A09: Security Logging and Monitoring Failures
10. A10: Server-Side Request Forgery

**The OWASP Top 10 Proactive Controls (OWASP, 2018)**

1. C1: Define security requirements
2. C2: Leverage security frameworks and libraries
3. C3: Secure database access
4. C4: Encode and escape data
5. C5: Validate all inputs
6. C6: Implement digital identity
7. C7: Enforce access controls
8. C8: Protect data everywhere
9. C9: Implement security logging and monitoring
10. C10: Handle all errors and exceptions

**What I have selected from the OWASP Top 10**

From the OWASP top 10 2021 I decided to focus on Injection because there lots of different types of injection ranging from command injection to SQL injection.

**What is an injection attack?**

IBM define injection attacks as a type of attack that allows an attacker to inject code into a program or query or inject malware onto a computer in order to execute remote commands that can read or modify a database, or change data on a web site. (IBM, N.D)

**What is an SQL Injection attack?**

IBM defines SQL Injection as an attack that takes advantage of the SQL syntax to inject commands that can read or modify a database, or compromise the meaning of the original SQL query. (IBM, N.D)

**How to prevent injection attacks**

The OWASP top 10 proactive controls suggests to escape data and validate all inputs in order to prevent all types of injection attacks (OWASP, 2018).

**How to prevent SQL Injection attacks**

OWASP suggests 4 different ways to defending against SQL injection option 1 is to use prepared statements, option 2 is to use properly constructed stored procedure, option 3 is to all list input validation and option 4 is to escape all user supplied input (OWASP, N.D).

**What is a prepared statement?**

Prepared statements are parameterized queries that force the developer to define all SQL code first and pass in each parameter to the query later.

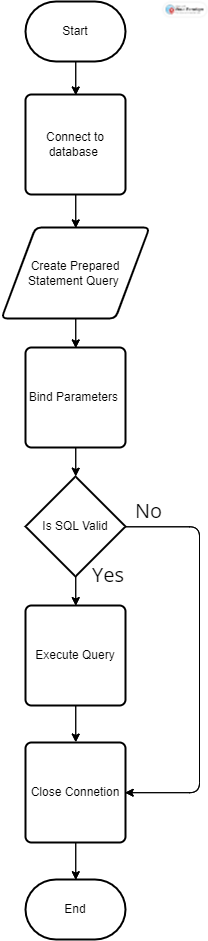
**Example of an SQL Injection query**

SELECT \* FROM Users WHERE UserId = 1 OR 1=1;

This query would return all rows for the user table as 1=1 is always true.

This table could contain confidential information such as the first name, last name and address information about all users for a particular service which a hacker could sell for financial gain or use to steal someone’s identity.

**SQL Injection Flow chart**

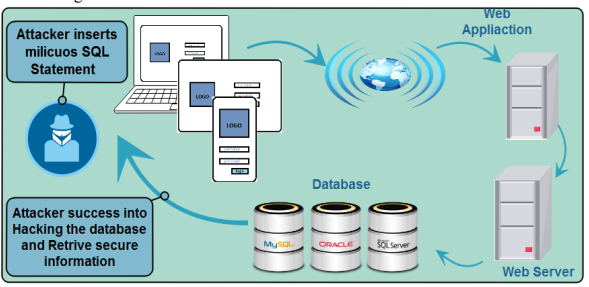


**Creating a prepared statement in Python**

import mysql.connector  
  
try:  
 connection = mysql.connector.connect(host='localhost',  
 database='python\_db',  
 user='admin',  
 password='root')  
  
 cursor = connection.cursor(prepared=True)   
 sql\_update\_query = "UPDATE Users SET Password = %s WHERE UserId = %s"  
  
 data\_tuple = ('qwerty', 1)  
 cursor.execute(sql\_update\_query, data\_tuple)  
 connection.commit()  
 print("User table updated using the prepared statement")  
  
except mysql.connector.Error as error:  
 print("parameterized query failed {}".format(error))  
finally:  
 if connection.is\_connected():  
 cursor.close()  
 connection.close()  
 print("MySQL connection is closed")

**Types of SQL Injection attacks**

The classic types of SQL injection attacks are Tautology based attacks, piggy-backed queries, logically incorrect attacks, union query, inference attack, stored procedure attack, blind injection, timing attacks, Compound SQL Injection attacks, Fast Flux SQL Injection attacks (P. Kumar and R. K. Pateriya, 2012)



**Reference List**

OWASP. (2021). OWASP Top 10:2021. Available from: <https://owasp.org/Top10/> [Accessed 16th March 2024]

OWASP. (2018). OWASP Proactive Controls. Available from: <https://owasp.org/www-project-proactive-controls/> [Accessed 16th March 2024]

OWASP. (N.D). SQL Injection Prevention Cheat Sheet. Available from: <https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html> [Accessed 16th March 2024]

IBM. (N.D). Injection attacks. Available from: <https://www.ibm.com/docs/en/snips/4.6.0?topic=categories-injection-attacks> [Accessed 16th March 2024]

P. Kumar and R. K. Pateriya. (2012) ‘A survey on SQL injection attacks, detection and prevention techniques’, *Third International Conference on Computing, Communication and Networking Technologies.* Coimbatore, India, 2012. America: IEE. pp. 1-5, doi: 10.1109/ICCCNT.2012.6396096.