

CLARITAS ACADEMICS

HSC Software Engineering

Comprehensive Course Notes, Architecture Guide & Master Examination Bank

CLARITY • CONFIDENCE • RESULTS

Syllabus & Examination Alignment

Curriculum: NSW Education Standards Authority (NESA) Software Engineering 11–12 (2022 Syllabus)

Course Level: Year 12 Higher School Certificate (HSC)

Modules Covered:

- **Module 1:** Programming for the Web
- **Module 2:** Software Automation
- **Module 3:** Secure Software Architecture
- **Module 4:** Software Engineering Project

Publication Release: 2026

How to use

Welcome to the **Claritas Academics Study Guide for HSC Software Engineering**.

This textbook is engineered specifically for the 2022 NESA syllabus. Rather than overwhelming you with abstract computer science theory, every page is structured around **Clarity, Confidence, and Results**—giving you the exact technical depth, code structures, and exam strategies required to achieve a Band 6. Throughout each chapter, you will encounter dedicated visual cues and pedagogical callout boxes designed to guide your revision:

Core Pedagogical Features

Syllabus Definition

What it is: The official, examiner-approved definition of a syllabus concept.

How to use it: Memorise these definitions verbatim. In the HSC written examination, 1-mark and 2-mark questions frequently award full marks simply for providing a precise, unambiguous definition of these terms.

Claritas Tip

What it is: High-yield exam techniques, structural writing frameworks, and shortcuts written by senior HSC examiners and tutors.

How to use it: Pay special attention to these when preparing for short-answer and extended-response questions. They explain how to structure your responses to guarantee maximum marks against NESA marking rubrics.

Common Trap

What it is: The most frequent misconceptions, logical bugs, and exam pitfalls where Year 12 students lose easy marks.

How to use it: Treat these as red flags. Before submitting any trial or HSC response, cross-check your work to ensure you haven't fallen into these common traps (e.g., confusing PWAs with hybrid native apps, or confusing authentication with authorisation).

 Industry Insight

What it is: Real-world case studies and architectural implementations from leading technology companies (e.g., Netflix, Uber, Stripe).

How to use it: In 5–8 mark extended response questions, referencing real-world engineering paradigms demonstrates high-level understanding and pushes your response from a Band 5 into the top of Band 6.

Structural & Visual Elements

- **Syllabus Dot Point Banners:** Every chapter begins with an explicit syllabus mapping blockquote. Cross-reference this with your school assessments to ensure complete curriculum alignment.
- **Architectural Flowcharts & Diagrams:** System designs, Class Diagrams, and Data Flow Diagrams (DFDs) are rendered in strict compliance with the official *NESA Software Engineering Course Specifications Document*.
- **Syntax-Highlighted Code Snippets:** All programming implementations (Python, SQL, HTML/CSS, and NESA Pseudocode) use consistent typographic formatting and indentation standards.

Assessment Architecture

At the conclusion of your study, you will encounter two tiers of assessment:

1. **Chapter Review Questions:** Targeted multiple-choice and short-answer questions situated at the end of each chapter to immediately verify conceptual understanding.
2. **Module Review Tests:** Comprehensive examination papers at the end of each module simulating the exact rigor, question distribution, and timing of the HSC paper.
3. **Full NESA-Style Marking Rubrics:** Every question is accompanied by an authentic criteria-to-marks matrix and exemplar Band 6 response, showing you exactly what markers look for.

Table of Contents

Module 1: Programming for the Web

CHAPTER	TITLE	PAGE
Chapter 1	Internet Architecture, Protocols & Web Security	1
Chapter 2	Front-End Architecture, Standards & Progressive Web Apps	11
Chapter 3	Front-End Frameworks, Tooling & Performance Engineering	24
Chapter 4	Back-End Engineering, Web Requests & Relational Databases	32

Module 2: Software Automation

CHAPTER	TITLE	PAGE
Chapter 5	Automation Paradigms, DevOps & the MLOps Lifecycle	45
Chapter 6	Machine Learning Algorithms & Training Models	55
Chapter 7	Neural Networks & Object-Oriented ML Architecture	70
Chapter 8	Sociotechnical Impacts, Human Behaviour & Algorithmic Bias	81

Module 3: Secure Software Architecture

CHAPTER	TITLE	PAGE
Chapter 9	Secure Design Principles, the SDLC & Cryptographic Defences	89
Chapter 10	Vulnerability Mitigation, User Controls & Secure Code	99
Chapter 11	Security Verification, Hardening & Incident Response	107
Chapter 12	Enterprise Collaboration, Ethics, Law & Societal Impacts	114

Module 4: Software Engineering Project

CHAPTER	TITLE	PAGE
Chapter 13	Project Inception, Feasibility & Development Approaches	121
Chapter 14	System and Data Modelling Tools	131
Chapter 15	Algorithm Design, Data Structures & Implementation	141
Chapter 16	Quality Assurance, Verification & Project Evaluation	153

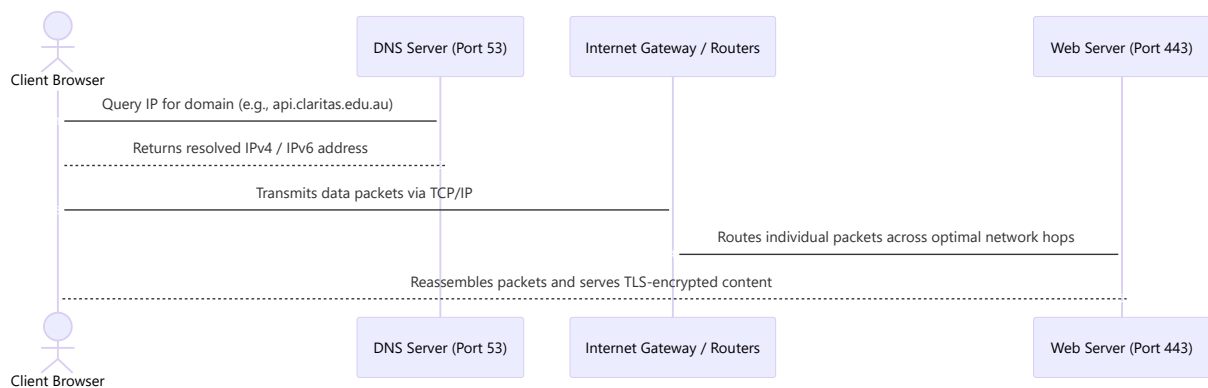
Chapter 1 - Internet Architecture, Protocols & Web Security

Syllabus Dot Points Covered:

- *Data transmission using the web* — Investigate and practise how data is transferred on the internet: data packets, internet protocol (IP) addresses, including IPv4, domain name systems (DNS)
- *Data transmission using the web* — Investigate and describe the function of web protocols and their ports: HTTP, HTTPS, TCP/IP, DNS, FTP, SFTP, SSL, TLS, SMTP, POP 3, IMAP
- *Data transmission using the web* — Explain the processes for securing the web: Secure Sockets Layer (SSL) certificates, encryption algorithms, encryption keys, plain text and cipher text, authentication and authorisation, hash values, digital signatures
- *Data transmission using the web* — Investigate the effect of big data on web architecture: data mining, metadata, streaming service management

1.1 The Mechanics of Internet Data Transmission

The internet is a global network of interconnected computers communicating via standardized protocol suites. Understanding how raw data moves between client devices and servers requires examining packet switching, addressing, and name resolution.



Data Packets and Packet Switching

Information transferred across the internet is never transmitted as a single, contiguous stream. Instead, files, media, and web payloads are decomposed at the transport layer into discrete chunks known as **data packets**.

Syllabus Definition

Data Packet: A formatted unit of data carried by a packet-switched network. A packet consists of three fundamental sections:

1. **Header:** Contains critical routing and sequencing metadata, including the Source IP address, Destination IP address, packet sequence number, and protocol identifier.
2. **Payload:** The actual chunk of user or application data being transmitted.
3. **Trailer (or Footer):** Contains end-of-packet markers and error-checking data, predominantly a Cyclic Redundancy Check (CRC) or checksum to verify packet integrity.

In **packet switching**, individual packets traverse intermediate networking hardware (switches and routers) independently. Packets may take entirely different physical routes depending on instantaneous network congestion, link failures, and cost metrics. Upon reaching the destination host, the network stack inspects the header sequence numbers, flags missing packets for retransmission (under TCP), and reassembles the payload into the original data structure.

Internet Addressing: IPv4 vs IPv6

Every node participating in internet communication requires a unique logical network identifier known as an **Internet Protocol (IP) Address**.

- **IPv4 (Internet Protocol version 4):** A 32-bit numeric addressing scheme expressed as four decimal octets separated by periods (e.g., `192.168.1.104` or `172.217.167.78`). Each octet represents 8 bits (values 0 to 255), yielding a theoretical ceiling of $2^{32} \approx 4.29$ billion distinct addresses. The exhaustion of unallocated IPv4 address blocks necessitated intermediate solutions like Network Address Translation (NAT) and Classless Inter-Domain Routing (CIDR).
- **IPv6 (Internet Protocol version 6):** Designed as the definitive replacement for IPv4, utilizing a 128-bit address space represented as eight groups of four hexadecimal digits separated by colons (e.g., `2001:0db8:85a3:0000:0000:8a2e:0370:7334`). IPv6 provides $2^{128} \approx 3.4 \times 10^{38}$ addresses, virtually eliminating address exhaustion, removing the need for NAT, and embedding native security features.

Domain Name System (DNS)

Humans identify internet resources through memorable alphabetic domain names (such as `nsw.gov.au`), whereas routing hardware operates purely on numerical IP addresses. The **Domain Name System (DNS)** serves as the distributed hierarchical directory service resolving these human-readable domain names into routable IP addresses. When a query is initiated, the client contacts a recursive resolver, traversing the root name servers, Top-Level Domain (TLD) servers (e.g., `.au`, `.com`), and authoritative name servers until the corresponding IP address record (an A record for IPv4, or AAAA record for IPv6) is retrieved and cached locally.

1.2 Web Protocols and Assigned Port Numbers

Network communication relies on strict protocols operating across different layers of abstraction.

Network interfaces use **virtual ports**—logical numeric endpoints within an operating system—to direct incoming network traffic to the appropriate application process.

PROTOCOL NAME	FULL NAME	STANDARD PORT(S)	PRIMARY PURPOSE & FUNCTIONAL MECHANICS
HTTP	Hypertext Transfer Protocol	80	Stateless application-layer protocol for fetching HTML, images, and API data. Transmits data as unencrypted plain text.
HTTPS	Hypertext Transfer Protocol Secure	443	HTTP layered over an encrypted TLS/SSL cryptographic tunnel, protecting data against eavesdropping and tampering.
TCP/IP	Transmission Control Protocol / Internet Protocol	N/A (<i>Stack</i>)	Fundamental suite. IP routes individual packets across network boundaries; TCP establishes a connection-oriented, reliable session guaranteeing ordered packet delivery.
DNS	Domain Name System	53	Translates human-friendly hostnames into machine-readable IP addresses using UDP (for fast queries) and TCP (for zone transfers).
FTP	File Transfer Protocol	20 (Data), 21 (Control)	Client-server protocol designed for raw file transfers between hosts. Transmits credentials and data in cleartext.
SFTP	SSH File Transfer Protocol	22	Secure alternative to FTP operating as an extension of the Secure Shell (SSH) protocol; provides encrypted file manipulation.

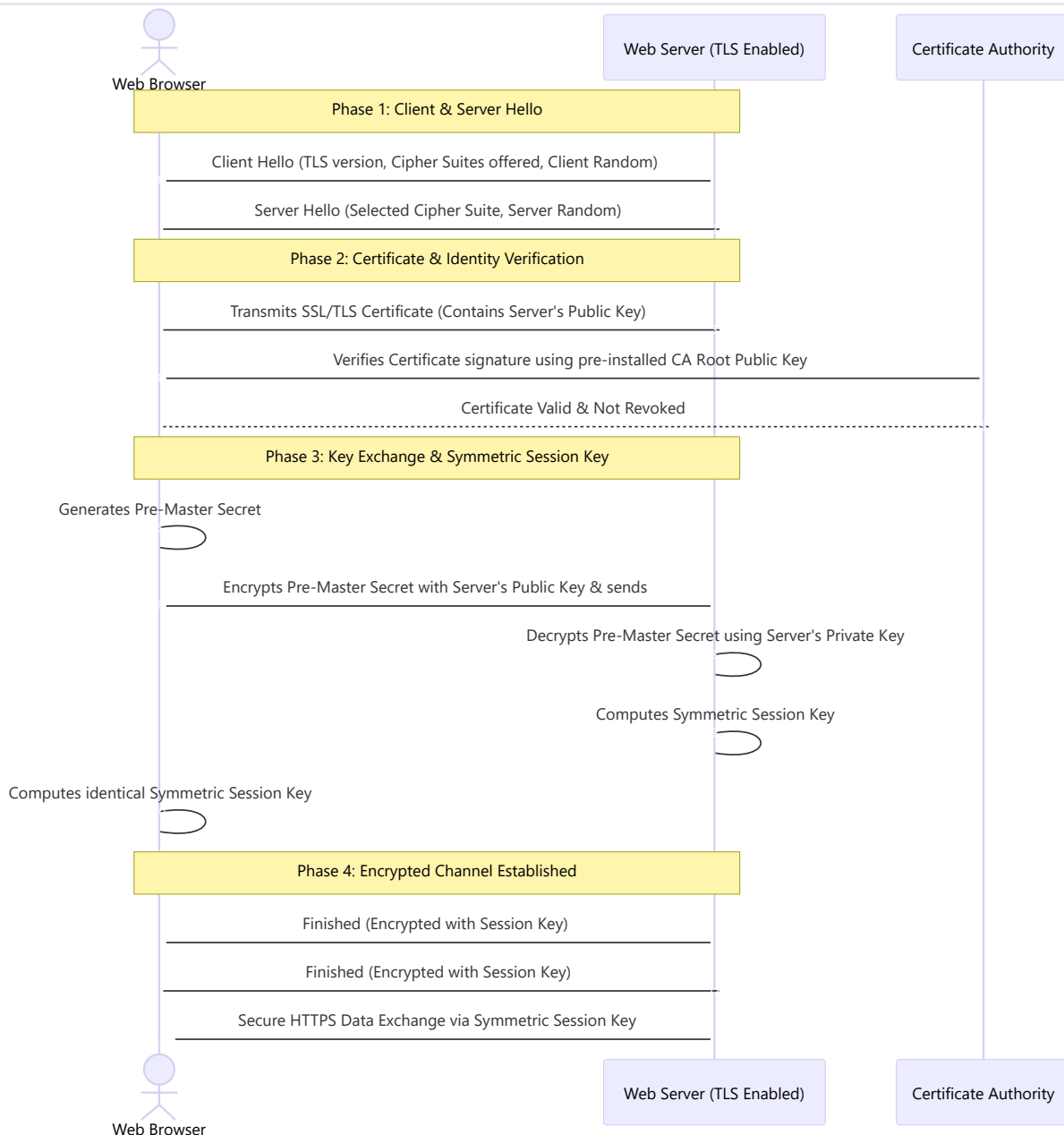
PROTOCOL NAME	FULL NAME	STANDARD PORT(S)	PRIMARY PURPOSE & FUNCTIONAL MECHANICS
SSL / TLS	Secure Sockets Layer / Transport Layer Security	443 (default for web)	Cryptographic protocols providing endpoint authentication, data integrity, and channel privacy across transport layers. TLS has replaced the legacy, deprecated SSL.
SMTP	Simple Mail Transfer Protocol	25 , 587	Push protocol used by email clients and mail transfer agents (MTAs) to send and relay outgoing messages across networks.
POP3	Post Office Protocol version 3	110 (Plain), 995 (SSL/TLS)	Pull protocol used by email clients to download messages from a server. Typically deletes messages from the server upon local receipt, tying mail history to a single device.
IMAP	Internet Message Access Protocol	143 (Plain), 993 (SSL/TLS)	Advanced pull protocol that synchronises email state bidirectionally across the server and multiple client devices. Messages remain stored centrally.

Claritas Exam Tip: POP₃ vs IMAP

In HSC examination questions comparing mail protocols, remember the core operational distinction: **POP₃** downloads messages locally and typically purges them from the remote server, meaning state is isolated to one device. **IMAP** maintains the master message archive and folder hierarchy directly on the mail server, synchronising changes across multiple client devices.

1.3 Securing Web Communications

Securing web applications requires establishing end-to-end data confidentiality, verifying identity, and validating that transmitted information has not suffered malicious tampering.

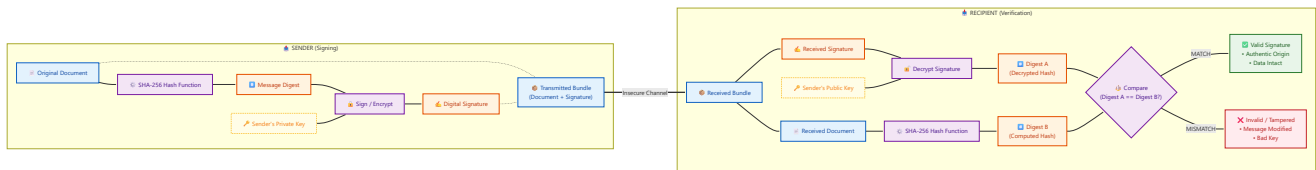


The Cryptographic Toolkit

- **Plain Text vs Cipher Text:** Plain text is unencrypted, human-readable data (e.g., an exposed credit card number string). Cipher text is the scrambled, unintelligible output generated by running plain text through an encryption algorithm parameterized by a cryptographic key.
- **Symmetric Encryption:** Uses a single, shared secret key for both data encryption and decryption (e.g., Advanced Encryption Standard [AES]). While computationally fast and ideal for encrypting large data payloads, it suffers from the "key distribution problem"—both parties must exchange the shared key securely without an eavesdropper intercepting it.
- **Asymmetric Encryption:** Uses a mathematically paired set of keys: a **Public Key** (openly published to the world for encrypting data) and a **Private Key** (kept strictly confidential by the host for decrypting incoming data). Common algorithms include RSA and Elliptic Curve Cryptography (ECC). Asymmetric encryption is computationally expensive, making it impractical for encrypting large continuous streams of data.
- **Hashing:** A cryptographic mathematical function that maps an arbitrary-length input string to a unique, fixed-bit length output known as a **hash value** or message digest (e.g., SHA-256). Hashing

is strictly a **one-way function** (it cannot be decrypted back to plain text) and is deterministic (identical inputs always yield identical digests). Cryptographic hashing is used for password storage (paired with unique cryptographic salts to defend against rainbow table attacks) and verifying message integrity.

- **Digital Signatures:** Provides non-repudiation, authenticity, and data integrity. The sender hashes the message, then encrypts that generated hash digest using their own **Private Key**. The recipient decrypts the signature using the sender's **Public Key** and independently computes the hash of the received message. If both hash values match exactly, the recipient is mathematically guaranteed that the message was created by the sender and was not modified in transit.



⚠️ HSC Trap: Hashing is NOT Encryption

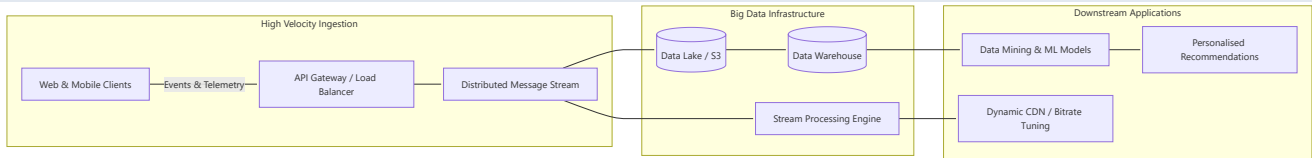
A classic student mistake in NESA exams is stating that "passwords are encrypted using SHA-256". Encryption is a two-way reversible process requiring a key to decrypt cipher text back into plain text. Hashing is an irreversible, one-way transformation. In secure software engineering, user passwords are **hashed** (and salted), never encrypted.

Authentication vs Authorisation

- **Authentication:** The process of verifying the claim of identity presented by a user or service (answering the question: *"Who are you?"*). Implemented via passwords, multi-factor authentication (MFA) tokens, biometrics, or cryptographic certificates.
- **Authorisation:** The process of determining the specific permissions, resources, and administrative actions an authenticated user is permitted to execute (answering the question: *"What are you allowed to do?"*). Implemented through Role-Based Access Control (RBAC) or Access Control Lists (ACLs).

1.4 Big Data and its Architectural Impact on the Web

Modern internet applications generate and consume vast quantities of data characterized by the 3Vs: **Volume** (terabytes to petabytes of logs and media), **Velocity** (millions of concurrent real-time transactions), and **Variety** (structured relational tables, semi-structured JSON, and unstructured video/audio).



Data Mining and Metadata

- **Metadata:** Structured "data about data" providing descriptive context, structural relationships, and administrative tracking (e.g., image EXIF tags, timestamp logs, content length, video encoding formats). In web architecture, rich metadata enables search engine indexing, asset caching rules, and precise routing.
- **Data Mining:** The automated computational process of extracting previously unknown patterns, correlations, anomalies, and predictive trends from massive datasets using machine learning and statistical models.

Streaming Service Management

Delivering high-definition, live or on-demand multimedia across unreliable global networks requires specialised architectural strategies:

1. **Content Delivery Networks (CDNs):** Geographically distributed networks of proxy edge servers caching static and streaming video fragments close to end users, minimizing latency and backhaul network saturation.
2. **Adaptive Bitrate Streaming (ABR):** Protocols (such as HLS or MPEG-DASH) that split media into small 2-to-10-second segments encoded at multiple resolution/bitrate tiers. Client-side logic monitors real-time network throughput and dynamically requests higher or lower bitrate chunks, preventing buffering or stream interruption.
3. **Data Sanitisation:** In big data logging pipelines, data sanitisation strips or masks Personally Identifiable Information (PII)—such as IP addresses, credit card numbers, and physical addresses—ensuring massive analytics databases comply with privacy regulations.

Chapter 1 Practice Questions

Multiple Choice Questions

Question 1.1 [Source: Claritas Original]

An internet user types `https://secure.bank.com.au` into their web browser. Which combination correctly identifies the protocol running over port 443 and the cryptographic method used to exchange the symmetric session key?

- (A) HTTP; Hashing with SHA-256
- (B) HTTPS; Asymmetric encryption using the server's public key
- (C) SFTP; Symmetric encryption using a pre-shared private key
- (D) DNS; Digital code signing via UDP

Question 1.2 [Source: Claritas Original]

A developer notices that an email client downloads incoming messages to a local workstation, deleting them from the remote server immediately upon receipt. Which protocol is the email client operating on?

- (A) IMAP
- (B) SMTP
- (C) POP3
- (D) SFTP

Short Answer Questions**Question 1.3 (3 marks)** [Source: Claritas Original]

A media streaming company broadcasts live international soccer matches. Explain how metadata and data mining are used within its web architecture to manage server load during peak demand periods.

Question 1.4 (4 marks) [Source: Claritas Original]

Distinguish between symmetric and asymmetric encryption. In your response, explain why modern HTTPS connections utilize both cryptographic approaches during a standard user session.

Chapter 1 Marking Guidelines & Exemplars**Question 1.1 Solution**

- **Correct Answer:** (B) HTTPS; Asymmetric encryption using the server's public key.
- **Rationale:** HTTPS operates over port 443. During the TLS handshake, asymmetric encryption (using the server's public key contained in its SSL/TLS certificate) is used to encrypt and securely transfer the pre-master secret/session key.

Question 1.2 Solution

- **Correct Answer:** (C) POP3.
- **Rationale:** Post Office Protocol 3 (POP3) pulls messages down to the client and deletes them from the central server by default, unlike IMAP which synchronises messages across multiple devices while keeping the master copies on the server.

Question 1.3 Marking Guide

MARKING CRITERIA	MARKS
• Comprehensively explains the role of both metadata and data mining in managing server load and predicting streaming demand. • Relates mechanisms directly to web architecture components (e.g. CDNs, dynamic resource allocation).	3
• Outlines the function of metadata and data mining with a basic connection to streaming or server load management.	2

MARKING CRITERIA	MARKS
Identifies a basic feature of metadata or data mining in isolation.	1

Band 6 Sample Exemplar:

Metadata provides descriptive data about the stream, such as video bitrates, client geographic locations, device screen resolutions, and connection speeds. Data mining analyzes large volumes of historical and real-time viewing logs to discover viewing patterns and predict geographic spikes in viewer traffic during specific match times. Web architects use these mined insights to balance server load by pre-caching stream segments on Edge Content Delivery Network (CDN) servers closest to anticipated high-density viewer clusters, dynamically scaling cloud server capacity to prevent buffering and network bottlenecks.

Question 1.4 Marking Guide

MARKING CRITERIA	MARKS
- Accurately distinguishes between symmetric and asymmetric encryption (key usage, computational overhead). - Fully explains the hybrid TLS architecture: why asymmetric encryption is used during the handshake and symmetric encryption for subsequent data payload transmission.	4
Explains the difference between symmetric and asymmetric encryption and outlines why both are used in HTTPS with minor omissions.	3
Describes symmetric and/or asymmetric encryption correctly without clearly explaining their combined use in HTTPS.	2
Identifies an isolated characteristic of encryption or keys.	1

Band 6 Sample Exemplar:

Symmetric encryption uses a single shared key to both encrypt and decrypt data, making it computationally fast but vulnerable to key interception during transmission. Asymmetric encryption uses a mathematically linked pair of keys—a public key for encryption and a private key for decryption—providing secure key exchange without requiring prior shared secrets, but at the cost of high computational and processing overhead.

HTTPS utilizes both in a hybrid system to maximize security and performance. During the initial TLS handshake, asymmetric encryption is employed: the browser uses the web server's verified public key to encrypt a randomly generated session key. The server decrypts it using its matching private key. Once this shared secret is established, the session transitions entirely to symmetric encryption,

allowing high-speed, secure transmission of the website's bulk data payloads without excessive CPU strain on the server or client.

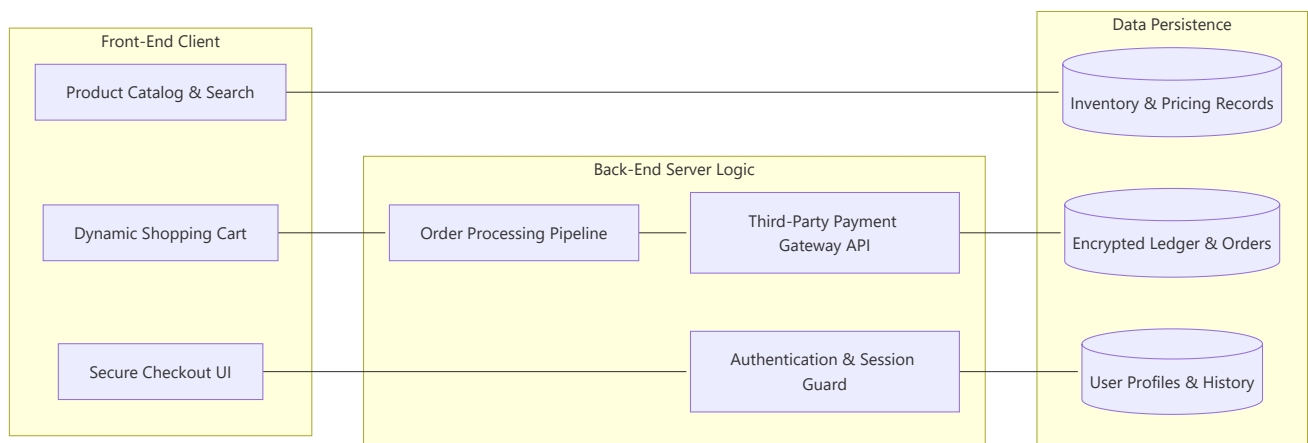
Chapter 2 - Front-End Architecture, Standards & Progressive Web Apps

Syllabus Dot Points Covered:

- *Data transmission using the web* — Explore the applications of web programming: interactive website/webpages, e-commerce, progressive web apps (PWAs)
- *Designing web applications* — Investigate and explain the role of the World Wide Web Consortium (W3C) in the development of applications for the web: Web Accessibility Initiative (WAI), internationalisation, web security, privacy, machine-readable data
- *Designing web applications* — Model elements that form a web development system: client-side (front-end) web programming
- *Designing web applications* — Explore and explain the influence of a web browser on web development, including the use of developer (dev) tools
- *Designing web applications* — Investigate cascading style sheets (CSS) and its impact on the design of a web application: consistency of appearance, flexibility with browsers or display devices, CSS maintenance tools
- *Designing web applications* — Design, develop and implement a progressive web app (PWA): the application of design and user interface (UI) and user experience (UX) principles of font, colour, audio, video and navigation; a UI that considers accessibility and inclusivity

2.0 Architectural Applications of Web Programming: E-Commerce Systems

Web programming facilitates sophisticated enterprise ecosystems, most notably **electronic commerce (e-commerce)** platforms that automate the commercial exchange of goods, services, and digital assets.

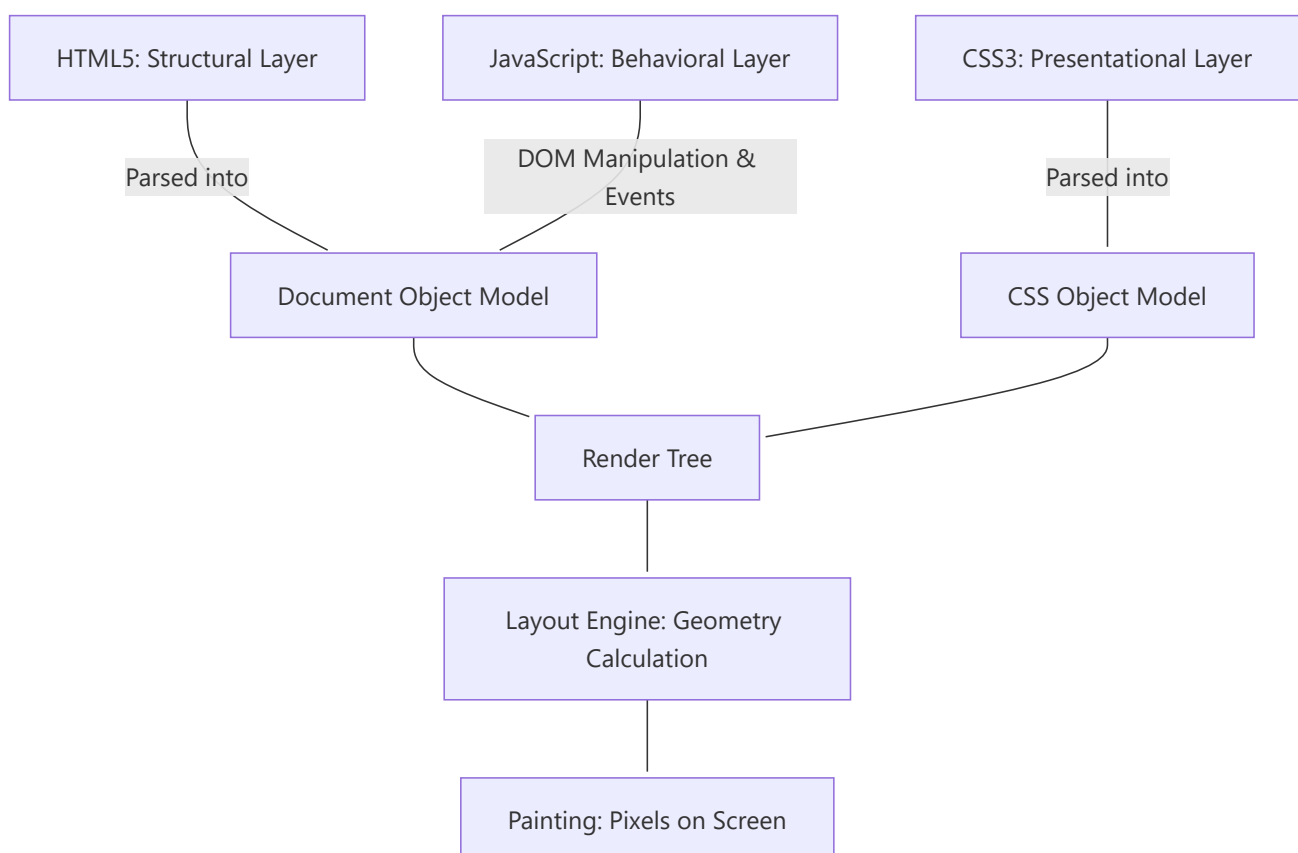


Core Functional Subsystems of E-Commerce Architectures

1. **Identity & Access Management:** Authenticates user sessions, secures credential stores, and preserves encrypted customer profiling (shipping addresses, historical invoices).
2. **Dynamic Product Catalogues:** Categorised, searchable collections presenting high-resolution media, dynamic stock counts, pricing tiers, and community review ratings.
3. **Stateful Shopping Cart:** Tracks selected item SKUs, variants (size, colour), and unit quantities across state changes using combinations of client storage (`LocalStorage`) and server-side session databases.
4. **Payment Processing Gateways:** Decouples sensitive credit card handling from the core application by tokenizing payment details and dispatching them securely to compliant PCI-DSS third-party services (e.g., Stripe, PayPal).
5. **Real-Time Order Fulfilment & Telemetry:** Dispatches transactional automated emails (via SMTP APIs), generates dynamic packing slips, and exposes live parcel tracking interfaces.

2.1 The Front-End Hierarchy: HTML, CSS, and JavaScript

Client-side web engineering operates entirely within the host execution environment of the user's web browser. It is structured into three distinct layers of abstraction:



1. **HTML (Hypertext Markup Language):** Provides semantic structure, marking up headers, paragraphs, forms, lists, and interactive media.
2. **CSS (Cascading Style Sheets):** Dictates presentation, visual styling, typography, responsive breakpoints, and UI layouts across various display devices.

3. **JavaScript (JS):** The client-side programming language that introduces behavioral logic, handles asynchronous networking (e.g., Fetch API), manages client state, and manipulates the Document Object Model (DOM) dynamically in response to user events.

Client: A device or software that requests services/resources from a server over a network.

Ex: Web browser, App, Desktop

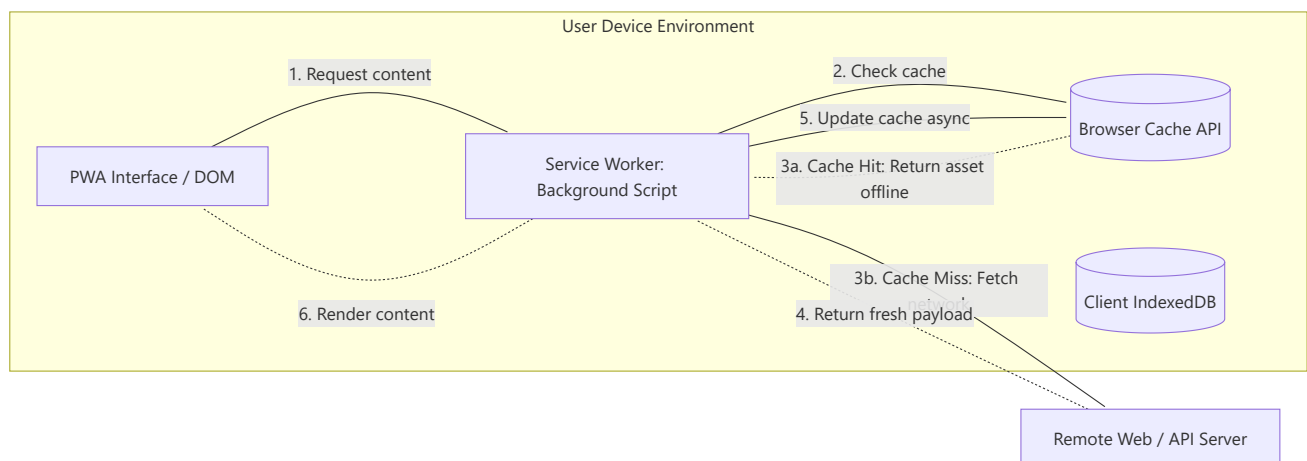
Server: A system that provides services, data or resources to clients. It processes client requests and responds.

Ex: Web Server, DB server, Game server

CLIENT SIDE	SERVER SIDE
Runs in the browser using JS	Runs on the web server (PHP, Python, Java)
Renders UI and handles user interactions within the browser.	Handles all the logic that must be secured, dynamic, and not exposed to the client.
Form Validation, animation, updating UI	Handling logins, retrieving from DB, processing forms

2.2 Progressive Web Apps (PWAs) Architecture

Traditional web development faced a sharp divide: standard web pages reached anyone with a browser but lacked offline capability, while native apps offered deep hardware integration and offline reliability but required platform-specific codebases (Swift for iOS, Kotlin for Android) and app store distribution approvals. **Progressive Web Apps (PWAs)** bridge this divide by leveraging modern browser APIs to deliver native app-like capabilities via standard web technologies.



Core Components of a PWA

A web application qualifies as a PWA when it implements three core technical features:

1. **Web App Manifest (`manifest.json`)**: A JSON configuration file providing metadata required for the browser to treat the site as an installable application. It defines the application's name, standalone launch mode (which strips browser URL bars and navigation controls), app icons, theme colors, and start URL.
2. **Service Worker**: A client-side JavaScript file that the browser runs in the background, completely detached from the web page's main UI thread. Acting as a programmable network proxy, the service worker intercepts every outgoing HTTP request made by the application. It can serve cached files instantly when offline, synchronize data in the background when connectivity resumes, and listen for push notifications.
3. **Secure Context (HTTPS)**: Because service workers intercept network traffic and manage background caching, browsers enforce that PWAs run exclusively over encrypted HTTPS connections to prevent Man-in-the-Middle (MitM) cache poisoning attacks.

```
{
  "name": "Claritas Field Engineer Portal",
  "short_name": "ClaritasApp",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#0052cc",
  "icons": [
    {
      "src": "/icons/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ]
}
```

Comparative Architecture: Traditional Websites vs Native Apps vs PWAs

ARCHITECTURAL FEATURE	TRADITIONAL WEBSITE	NATIVE MOBILE APPLICATION	PROGRESSIVE WEB APP (PWA)
Execution Environment	Web Browser Tab	Operating System directly (Objective-C/Swift/Kotlin)	Browser Engine running in standalone container
Offline Capability	None (Fails with network loss)	Full native local database access	Full (Managed by Service Worker & Cache API)
Distribution & Installation	Hyperlink URL; No installation	App Stores (Requires review, download, packaging)	Web URL installation directly to Home Screen

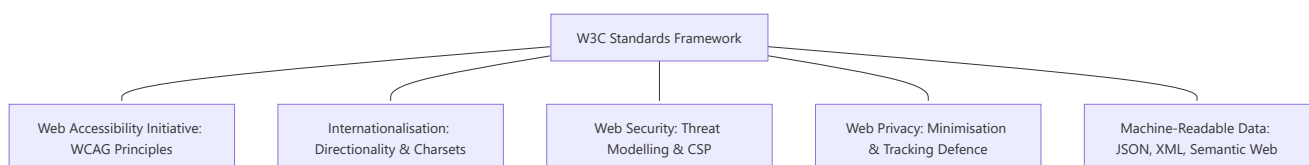
ARCHITECTURAL FEATURE	TRADITIONAL WEBSITE	NATIVE MOBILE APPLICATION	PROGRESSIVE WEB APP (PWA)
Hardware Access	Highly restricted sandbox	Unrestricted access to hardware sensors, Bluetooth	Moderate-to-high via modern Web APIs (Camera, GPS)
Codebase Overhead	Single web codebase	Multiple distinct platform-specific codebases	Single unified web codebase across all devices

☰ 🖥️ Industry Case Study: Uber PWA

Uber engineered m.uber.com as a Progressive Web App designed specifically to enable ride booking on low-end 2G mobile networks. By isolating its core ride-hailing functionality into a lightweight PWA shell, the entire initial application core compresses to under 50KB gzip. The service worker caches ride-request layouts, allowing near-instantaneous load times on low-bandwidth smartphones without requiring users to download a 100MB+ native app store package.

2.3 W3C Standards: Inclusivity, Privacy, and Structure

The **World Wide Web Consortium (W3C)** is the international standards organization responsible for establishing long-term technical specifications that govern the web. Rather than letting individual browser manufacturers invent conflicting, proprietary technologies, the W3C enforces uniform standards across five critical domains:



Web Accessibility Initiative (WAI)

The W3C's Web Accessibility Initiative creates the **Web Content Accessibility Guidelines (WCAG)**, built upon four core principles (POUR):

- **Perceivable:** Information and UI components must be presentable to users in ways they can perceive (e.g., providing comprehensive `alt` attributes for screen readers to translate images to audio, maintaining minimum 4.5:1 text-to-background contrast ratios for low-vision users).
- **Operable:** UI navigation must be entirely operable via hardware keyboards alone, without requiring a mouse. Focus indicators must be visible, and interactive elements must not cause seizures

(avoiding rapid flickering).

- **Understandable:** Content must be readable with predictable layout behaviors, providing input validation and friendly error messages when forms are improperly completed.
- **Robust:** HTML markup must be completely valid and semantic, ensuring assistive technologies (such as screen magnifiers and screen readers) can parse DOM tree nodes reliably across changing platforms.

Internationalisation (I18N)

Web solutions must cater to multilingual and global user bases without requiring code redesigns. W3C guidelines mandate:

- Standardizing on UTF-8 character encoding to support non-Latin scripts (Arabic, Cyrillic, Kanji).
- Implementing CSS directionality rules (e.g., `dir="rtl"` for Right-to-Left reading orders like Hebrew and Arabic).
- Designing elastic layout containers that adapt when text expands across translations (e.g., German translations typically require 30% more visual width than English equivalents).

Machine-Readable Data

Web services frequently communicate without human intervention. Machine-readable formats—such as JSON, XML, and structured CSV—allow systems to parse, validate, and store information directly. The W3C establishes guidelines for metadata exchange and semantic markup (e.g., Schema.org microdata), enabling search engines to catalog web data automatically.

W3C Web Security and Privacy Frameworks

Beyond markup syntax, the W3C engineers concrete technical standards defending application integrity and safeguarding user privacy:

1. W3C Security Standards:

- **Threat Modelling:** A systematic engineering methodology used during the architectural phase to map potential system attack vectors (e.g., spoofing, information disclosure, privilege elevation) and institute defensive controls.
- **Web Application Security:** Standardised browser sandboxing protocols, including **Content Security Policy (CSP)** HTTP headers that restrict the domains from which scripts, styles, and workers can be fetched.
- **Cross-Site Scripting (XSS) Defence:** W3C guidelines govern how browsers isolate DOM execution contexts. XSS occurs when an attacker injects arbitrary executable JavaScript into an application because user input was rendered without sanitisation or encoding:

```
// VULNERABLE TO XSS: Direct HTML injection of unvalidated user input
document.getElementById("output").innerHTML = userInput;

// SECURE: Browser treats input strictly as data, neutralizing malicious scripts
document.getElementById("output").textContent = userInput;
```

- **Web Authentication (WebAuthn):** A specification enabling unphishable, hardware-backed, public-key authentication using local biometrics (TouchID, FaceID) or FIDO2 physical security keys, removing password transmission risks entirely.
- **Federated Identity:** W3C-backed single sign-on protocols (e.g., OAuth 2.0 / OpenID Connect) allowing users to authenticate securely through an established identity provider ("Sign in with Google") without exposing raw passwords to third parties.
- **Web Payment Security:** Standardised, browser-native payment request APIs that tokenize payment credentials securely within hardware sandboxes prior to network transmission.

2. W3C Privacy Standards:

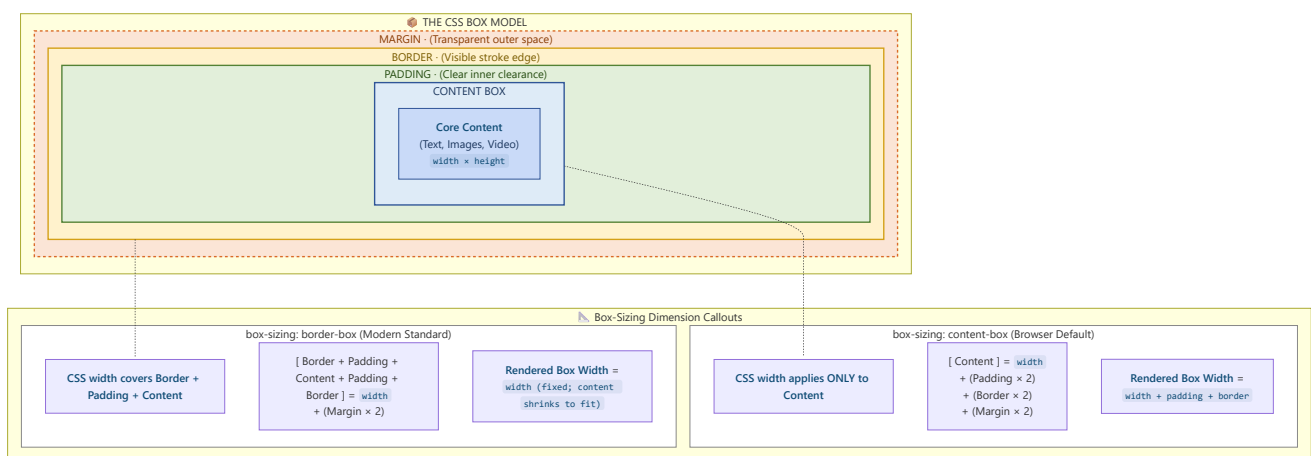
- **Privacy by Design:** Mandates that privacy safeguards are baked into core architectural schemas rather than patched post-deployment.
- **Data Minimisation:** Restricting data ingestion exclusively to fields strictly required for transaction processing, immediately discarding temporary telemetry.
- **Privacy Incubation:** Collaborative community testing of privacy-preserving APIs (such as anti-fingerprinting techniques and cross-site cookie restrictions) to halt tracking across independent internet domains.

2.4 Cascading Style Sheets (CSS) and the Box Model

CSS decouples page structure (HTML) from visual appearance, ensuring design consistency and enabling responsive adaptation across disparate display viewports.

The CSS Box Model

Every element rendered onto a web page is treated by the browser layout engine as a rectangular box comprising four concentric geometric zones:



- **Content:** The innermost area where text, images, and nested elements appear.
- **Padding:** The clear transparent clearance surrounding the content, inside the border.
- **Border:** A stroke line wrapping around the padding and content.
- **Margin:** The outermost clear area separating this element's bounding border from adjoining elements on the page.

CSS Syntax & Selection (NESA Standard Specification)

As specified in the HSC Software Engineering Course Specifications, CSS rules follow a strict grammar:

```
/* Comment */  
selector {  
    property: value;  
}
```

- **Comments:** Enclosed within `/*` and `*/`.
- **Selector:** Targets a specific HTML tag name (e.g., `h1`), an identifier (e.g., `#welcome`), or a reusable class (e.g., `.red-text`).
- **Property:** The styling attribute to modify (e.g., `font-size`, `color`, `background-color`).
- **Value:** The assigned configuration (e.g., `18px`, `red`, `rgb(0, 0, 100)`).

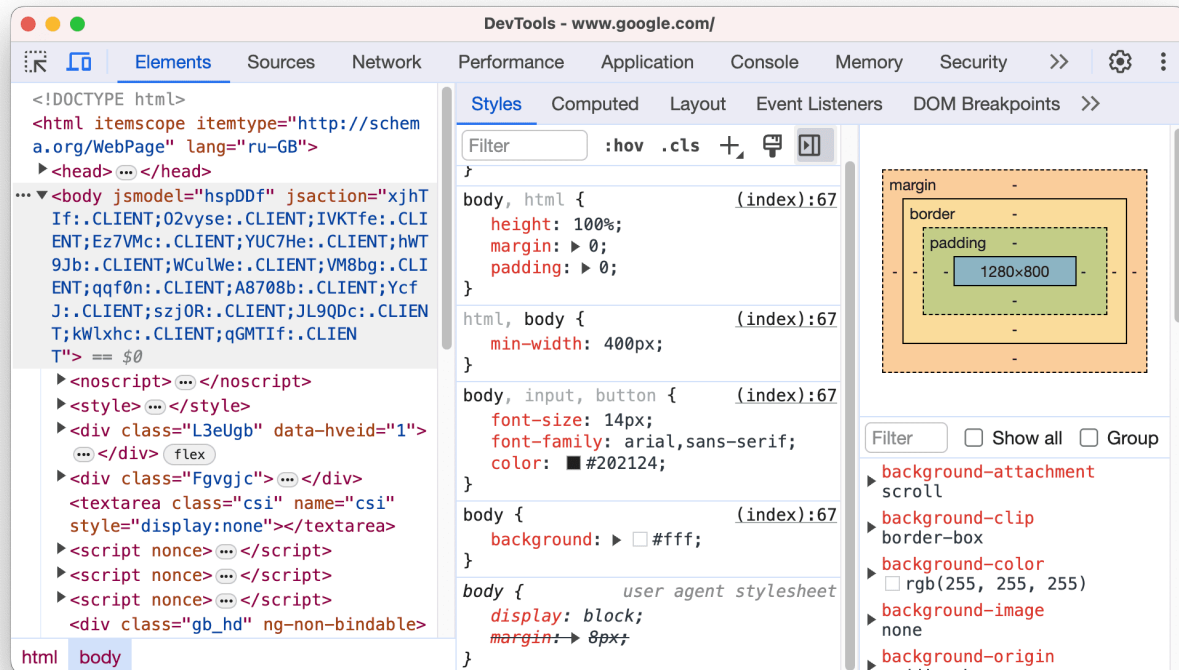
CSS Maintenance Tools

As web applications scale to hundreds of components, raw CSS files can become difficult to maintain, leading to specificity conflicts and code bloat. Software engineers use modern maintenance solutions:

1. **Preprocessors (e.g., Sass, LESS):** Extend CSS syntax with programmatic capabilities, including variables, nested selector trees, mathematical operations, and reusable code blocks known as **mixins**. These are compiled down to standard CSS for browser consumption.
2. **CSS Methodologies (e.g., BEM - Block, Element, Modifier):** A naming standard (e.g., `.card__button-disabled`) that enforces modularity and eliminates cascade inheritance collisions in enterprise development teams.

2.5 Browser Developer Tools (DevTools)

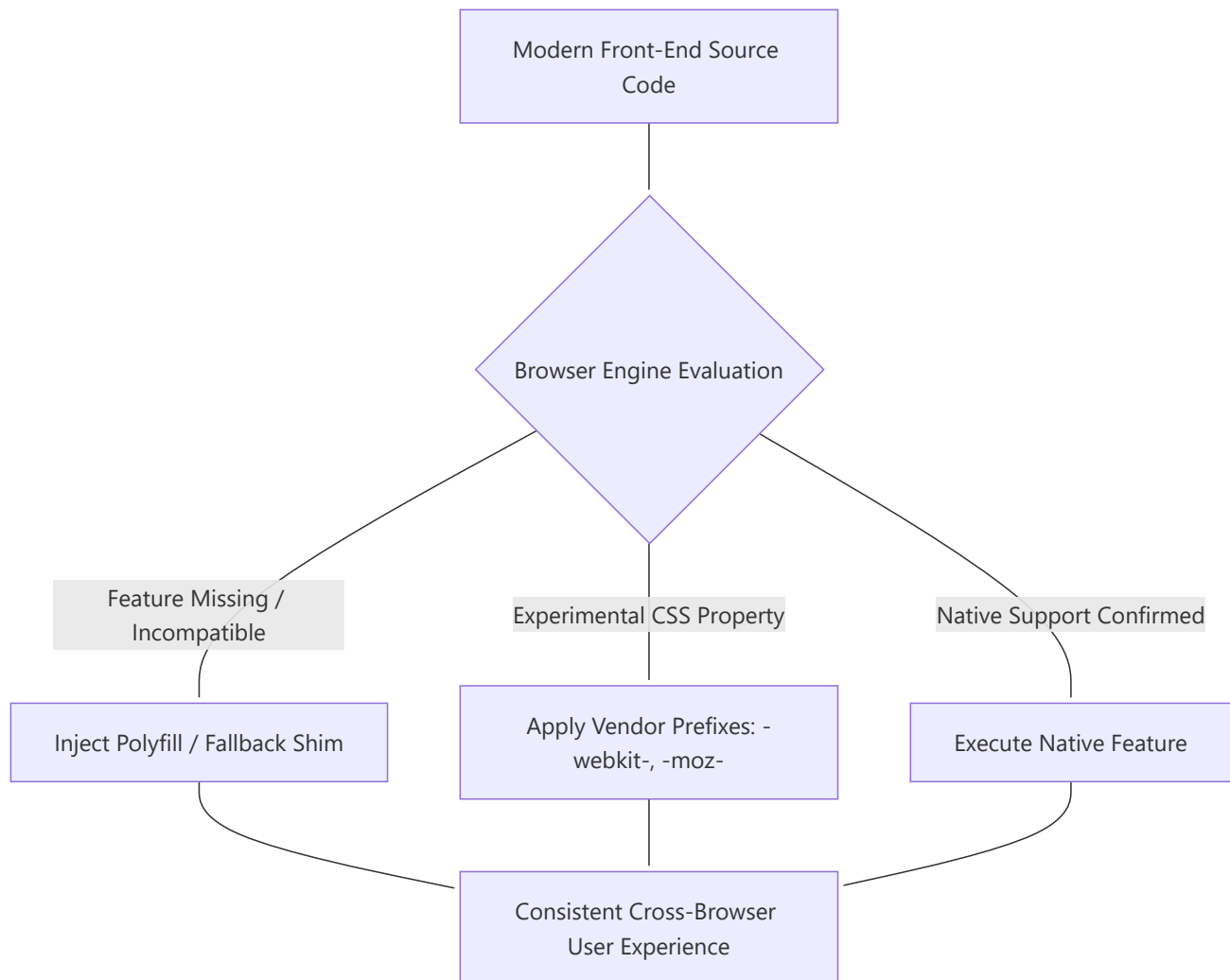
Built directly into all modern rendering engines, browser developer tools provide engineers with an integrated environment for inspecting, debugging, and profiling web code in real time:



- **Elements / Inspector Tab:** Allows live modification of DOM elements and CSS box model properties, displaying computed layout geometry instantly.
- **Console Tab:** Displays runtime JavaScript exceptions, warnings, and log outputs, serving as a live JavaScript REPL (Read-Eval-Print Loop) environment.
- **Network Tab:** Chronologically charts every network request dispatched by the page, detailing HTTP status codes (e.g., 200 OK, 404 Not Found), request/response headers, payload sizes, and waterfall timings to identify blocking scripts.
- **Application Tab:** Inspects and clears client-side persistent storage, including LocalStorage, SessionStorage, IndexedDB records, Cookies, and registered Service Worker lifecycles.
- **Lighthouse / Audit Tab:** Executes automated audits testing page performance, Web Accessibility (WAI compliance), SEO hygiene, and PWA installation criteria.

2.6 The Browser Runtime Environment & Cross-Browser Engineering

Web browsers act as the client runtime environment. However, different browsers rely on different underlying **rendering engines** (e.g., Blink for Chrome/Edge, Gecko for Firefox, WebKit for Safari). Because these engines interpret markup, layout models, and JavaScript APIs with minor divergences, software engineers implement defensive compatibility workflows:



Cross-Browser Compatibility Mechanics

- **CSS Vendor Prefixes:** Before bleeding-edge styling features are fully standardized by the W3C, browser vendors implement experimental variants requiring prefixes:
 - `-webkit-` (Chrome, Safari, iOS browsers, newer Edge)
 - `-moz-` (Mozilla Firefox)
- **Polyfills and Shims:** A polyfill is a supplementary JavaScript snippet that recreates modern browser API functionality in older runtime engines that lack native support (e.g., retrofitting the modern `fetch()` API into legacy browsers that only support raw `XMLHttpRequest`).
- **Graceful Fallbacks:** A design philosophy where a web component provides a reliable, simpler alternative when an advanced feature fails or is absent (e.g., serving a flat PNG image if interactive WebGL rendering is unsupported by the client GPU).
- **Feature Detection vs User-Agent Sniffing:**
 - **User-Agent Sniffing (Anti-Pattern):** Inspecting the browser's identifying metadata string (`navigator.userAgent`) to guess the browser brand and version. This is notoriously unreliable because strings can be spoofed or misreported.
 - **Feature Detection (Best Practice):** Directly testing programmatically whether a specific object or method exists in the runtime environment before invoking it:

```
// Robust Feature Detection for the Geolocation API
if ("geolocation" in navigator) {
    navigator.geolocation.getCurrentPosition(successHandler, errorHandler);
} else {
    // Graceful fallback for unsupported execution environments
    displayManualLocationInput();
}
```

Chapter 2 Practice Questions

Multiple Choice Questions

Question 2.1 [Source: 2025 CSSA Trial, Q1]

Which of the following best describes a Progressive Web App (PWA)?

- (A) A type of website with no offline capabilities
- (B) A website that only supports desktop browsers
- (C) A mobile application that requires installation from an app store
- (D) A web app that uses modern web capabilities to deliver an app-like experience

Question 2.2 [Source: 2025 HSC Exam, Q7]

Consider the following cascading style sheet (CSS) code fragment for a website:

```
body {
    font-size: 100%;
    font-style: italic;
    line-height: 1.8;
    font-weight: normal;
}
h1 {
    text-align: center;
    font-style: normal;
    font-size: 250%;
    letter-spacing: 5px;
    background-color: rgb(0, 0, 100);
    color: white;
    font-weight: 900;
    border: 3px solid green;
}
```

Which of the following best describes the appearance of the heading on the website?

- (A) White background, black text, green border
- (B) Centre aligned, italic, increased letter spacing
- (C) Blue background, white text, border around the heading
- (D) Left aligned, bold, and 2.5 times larger than the rest of the page

Short Answer Questions

Question 2.3 (3 marks) [Source: 2025 HSC Exam, Q16a]

A weather app is to be designed to display real-time updates such as rain commencing within 10 minutes and localized event cancellations. Compare the use of a Progressive Web App (PWA) with an interactive website for delivering this weather app.

Question 2.4 (3 marks) [Source: Sample Trial 2025, Q19b]

Explain how accessibility considerations improve the usability of web forms for a wide range of users, including those with disabilities.

Chapter 2 Marking Guidelines & Exemplars

Question 2.1 Solution

- **Correct Answer:** (D) A web app that uses modern web capabilities to deliver an app-like experience.
- **Rationale:** PWAs utilize web technologies (HTML, CSS, JS, Service Workers) to provide app-like capabilities, including offline access and home screen installation, without app store distribution.

Question 2.2 Solution

- **Correct Answer:** (C) Blue background, white text, border around the heading.
- **Rationale:** The `h1` rule declares `background-color: rgb(0, 0, 100)` (dark blue), `color: white`, and `border: 3px solid green`. It also overrides the body's italic style with `font-style: normal`.

Question 2.3 Marking Guide

MARKING CRITERIA	MARKS
• Provides a balanced comparison between a PWA and an interactive website, explicitly evaluating offline access, caching, and delivery of real-time weather updates. • Identifies architectural tradeoffs (e.g., connectivity dependence vs cached stale data).	3
• Outlines characteristics of a PWA and an interactive website with minor comparative links to the scenario.	2
• States an isolated feature of either a PWA or an interactive website.	1

Band 6 Sample Exemplar:

Both solutions offer cross-platform accessibility across mobile and desktop browsers via a single codebase. An interactive website requires a continuous internet connection to fetch and display dynamic radar data; if the user enters a cellular blackspot, the application fails completely.

In contrast, a PWA utilizes background service workers and local caching to deliver a resilient app-shell interface even when completely offline, allowing users to view previously loaded weather warnings. However, the PWA may display stale cached data until reconnected, whereas an interactive website guarantees that any information successfully loaded is up to the minute. Furthermore, the PWA can be installed directly to the user's home screen and receive push alerts for emergency weather events, offering a superior native app-like user experience.

Question 2.4 Marking Guide

MARKING CRITERIA	MARKS
• Comprehensively explains how specific accessibility implementations (e.g., semantic labeling, contrast, keyboard navigation) benefit users with disabilities as well as general users.	3
• Outlines several accessibility considerations and connects them to user interface usability.	2
• Identifies an isolated accessibility feature without linking to broad usability benefits.	1

Band 6 Sample Exemplar:

Adhering to W3C accessibility guidelines improves web form usability across multiple dimensions. Implementing semantic HTML `<label>` elements linked to form inputs enables screen-reading software to vocalise field purposes to visually impaired users, while simultaneously providing larger touch-target click areas for individuals with motor impairments or mobile users.

Establishing high-contrast text and explicit focus indicators allows visually impaired or color-blind users to navigate fields easily using a keyboard alone, eliminating reliance on a mouse. Clear inline error validation provides immediate, plain-language corrective prompts, ensuring users with cognitive disabilities or those experiencing situational stress can submit forms without confusion.

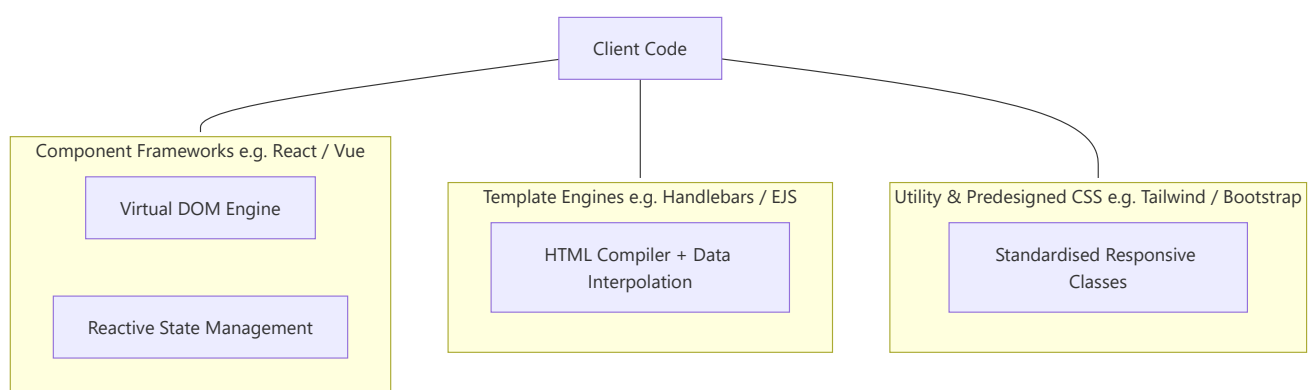
Chapter 3 - Front-End Frameworks, Tooling & Performance Engineering

Syllabus Dot Points Covered:

- *Designing web applications* — Explore the types and significance of code libraries for front-end web development: frameworks that control complex web applications, template engines, predefined CSS classes
- *Designing web applications* — Explain the use and development of open-source software in relation to web development
- *Designing web applications* — Investigate methods to support and manage the load times of web pages/applications
- *Designing web applications* — Research, experiment with and evaluate the prevalence and use of web content management systems (CMS)
- *Designing web applications* — Investigate the reasons for version control and apply it when developing web application

3.1 Front-End Libraries, Frameworks, and Engines

As web applications expanded from static informational pages to dynamic, reactive enterprise portals, raw JavaScript DOM manipulation became cumbersome and error-prone. Modern front-end architectures utilize specialized abstraction layers:



Component-Based Frameworks

Modern frameworks (such as React, Vue.js, and Angular) structure web interfaces into independent, reusable, and self-contained **components**. Rather than making slow, direct updates to the browser's physical DOM, frameworks like React maintain an in-memory **Virtual DOM**. When application state changes, the framework calculates the minimal difference (diffing) between the virtual and physical DOM, applying batch updates with minimal reflow and repaint cycles.

Template Engines

Template engines (such as EJS, Handlebars, and Pug) allow software engineers to interpolate dynamic server data directly into HTML templates using special delimiter syntax (e.g., `{{ username }}`). This separates raw application logic from visual layout structures, streamlining server-side rendering (SSR) pipelines.

Predesigned and Utility-First CSS Classes

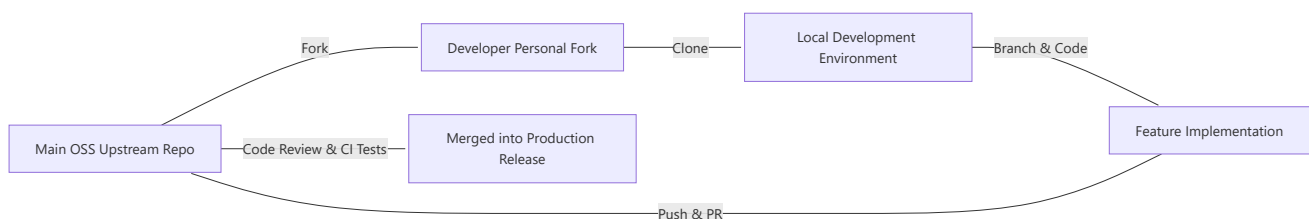
Writing custom CSS from scratch for every UI button or layout container often leads to inconsistent spacing and duplicate rules.

- **Component CSS Frameworks (e.g., Bootstrap):** Offer pre-styled UI components (modals, navbars, cards) out of the box, speeding up rapid prototyping.
- **Utility-First Frameworks (e.g., Tailwind CSS):** Provide granular, single-purpose utility classes (e.g., `flex`, `pt-4`, `text-center`, `bg-blue-500`) applied directly within HTML markup, enforcing design-token consistency and keeping final CSS bundle sizes small.

3.2 Open-Source Software (OSS) Ecosystems

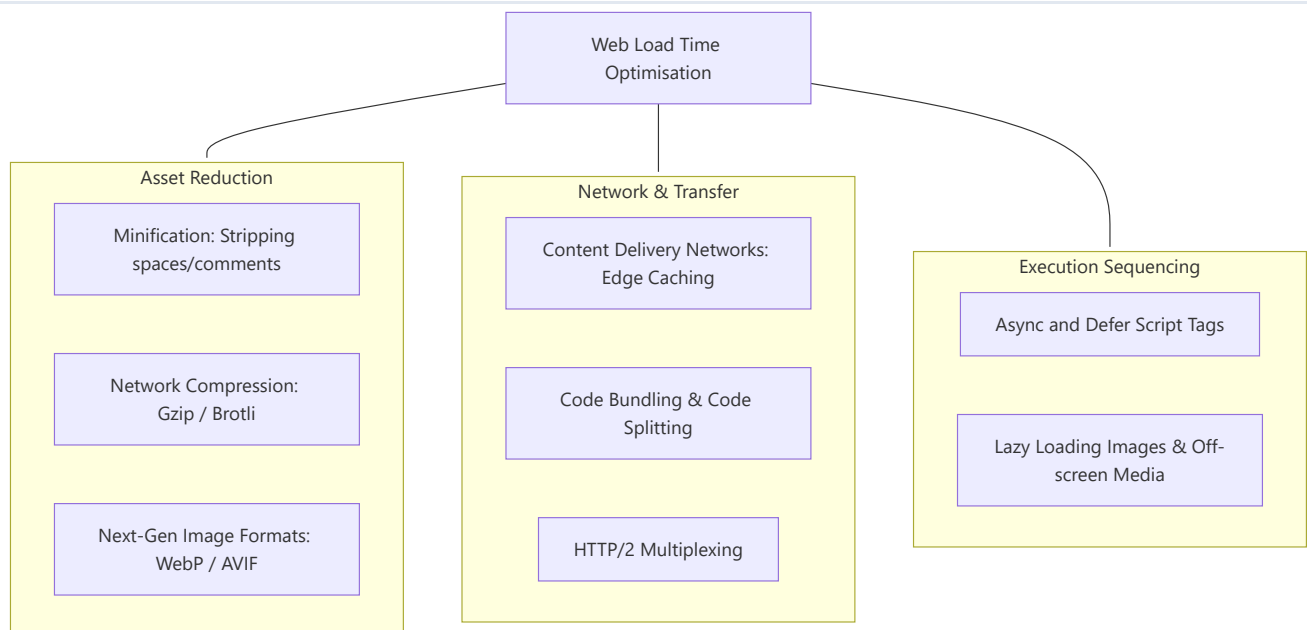
Modern web engineering relies heavily on the **Open-Source Software (OSS)** model, where application source code is made publicly available for inspection, modification, and collaborative improvement.

- **Development Workflow:** OSS platforms (predominantly GitHub and GitLab) rely on **Pull Requests (PRs)** and issue trackers. Engineers fork a shared repository, make modifications in an isolated branch, run automated tests, and submit a PR for code review by project maintainers.
- **Licensing Models:** Open-source projects include specific software licenses. Permissive licenses (such as the **MIT License**) grant developers unrestricted freedom to use, modify, and distribute code commercially without open-sourcing their proprietary derivative works. Copyleft licenses (such as the **GNU General Public License [GPL]**) legally mandate that any software using or extending the code must also release its full source code openly under the same license terms.



3.3 Web Performance: Managing Page Load Times

Page load latency directly impacts user bounce rates, conversion metrics, and search engine optimization (SEO) algorithms. Web performance engineering targets the **Critical Rendering Path** using several key strategies:



Performance Optimisation Techniques

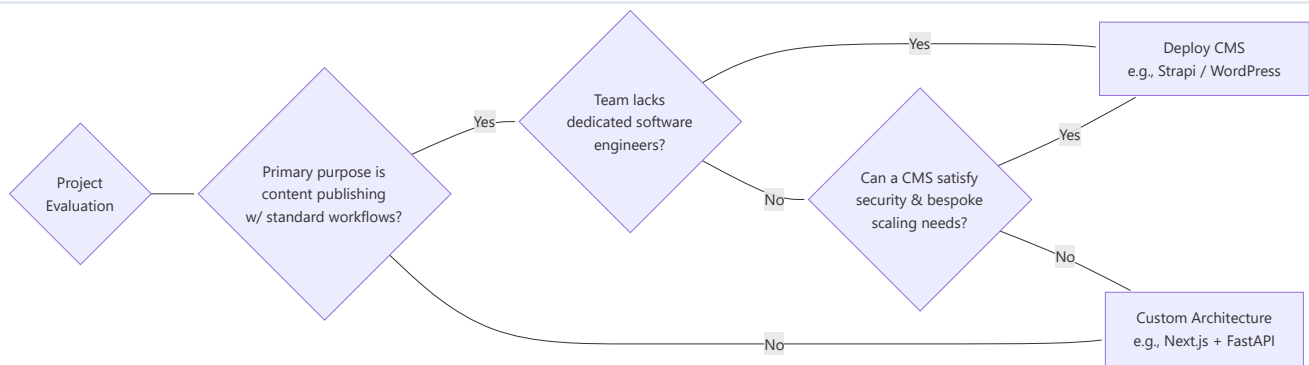
1. **Minification and Bundling:** Minification strips white space, line breaks, and developer comments from HTML, CSS, and JS files, and shortens variable identifiers (e.g., changing `userAuthenticationToken` to `a`). Bundling tools combine dozens of modular script files into single consolidated payloads, reducing the total number of round-trip HTTP requests.
2. **Next-Generation Image Formats & Resizing:** Large bitmap images are the primary cause of bloated web pages. Using modern formats like WebP or AVIF achieves 30–50% smaller file sizes than legacy JPEGs at equivalent visual quality. Responsive images (`<img srcset="...">`) ensure mobile devices download resolution-appropriate files rather than downscaling full 4K desktop graphics.
3. **Lazy Loading:** Defers the loading of off-screen images, videos, and non-critical scripts until the user scrolls them into the active browser viewport (`<img loading="lazy">`).
4. **Asynchronous JavaScript Loading:** By default, HTML parsers pause DOM construction whenever they encounter a `<script>` tag. Adding the `async` or `defer` attributes ensures external scripts download in the background without blocking page rendering:

```
<!-- Defer guarantees non-blocking download and executes in DOM order after parsing -->
<script src="analytics.js" defer></script>
```

5. **Browser and CDN Caching:** HTTP cache headers (`Cache-Control: max-age=31536000, immutable`) instruct the client's local browser and intermediate Content Delivery Network (CDN) edge servers to store static assets locally. Repeat visits load instantly from local memory cache rather than making redundant network requests to the origin server.

3.4 Content Management Systems (CMS) vs Custom Development

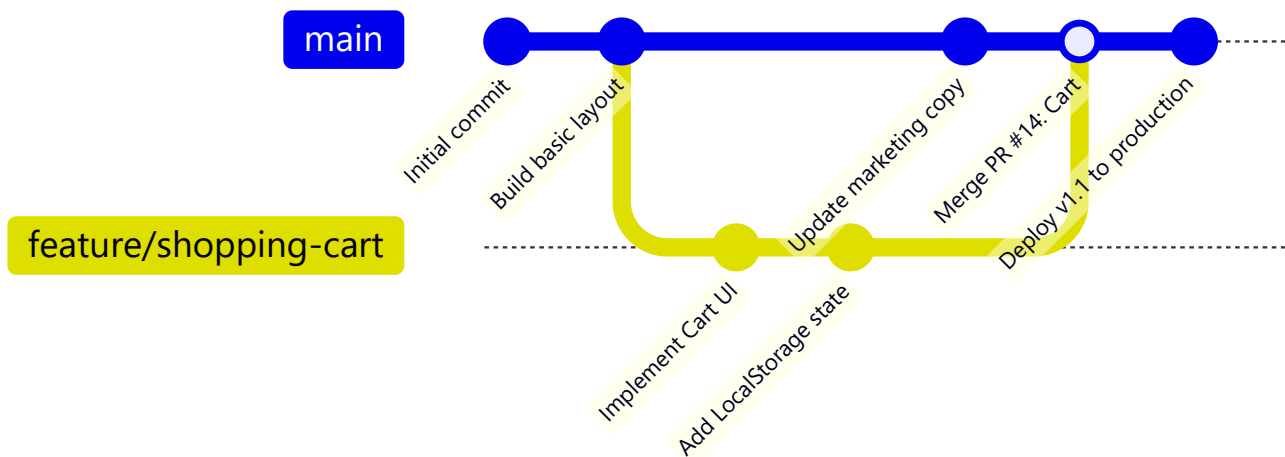
When commissioning web software, organizations must choose between leveraging an established **Web Content Management System (CMS)** or building a **Custom Web Application**.



CRITERION	CONTENT MANAGEMENT SYSTEM (CMS)	CUSTOM ENGINEERED WEB SOLUTION
Technical Barrier to Entry	Low; Intuitive GUI, non-technical authors publish content using WYSIWYG editors.	High; Requires professional software engineers proficient in front-end, back-end, and database stacks.
Time to Market	Extremely rapid; Pre-designed commercial themes and modular plugins provide instant functionality.	Slower initial deployment; System architecture, security layers, and UI components are coded from scratch.
Architectural Customisation	Constrained by the core CMS architecture; Complex bespoke workflows often require brittle third-party plugins.	Fully unconstrained; Can be tailored to precise business logic, bespoke algorithms, and custom relational schemas.
Performance Overhead	Often bloated; Generic CMS templates load unused CSS/JS scripts and run heavy database overhead.	Highly optimised; Ships minimal, bespoke code bundles engineered specifically for the target problem.
Security Surface Area	High maintenance; Ubiquitous platforms (like WordPress) are frequent targets of automated bot exploits targeting out-of-date plugins.	Controlled; Smaller attack footprint, though development teams are directly responsible for secure coding standards.

3.5 Version Control Systems (Git) in Web Development

Web engineering is inherently collaborative. **Version Control Systems (VCS)**, specifically distributed tools like **Git**, track changes to the project codebase over time, enabling teams to build features concurrently without overwriting each other's code.



Essential Version Control Mechanics

- **Repository (Repo):** The central database tracking all project files alongside their historical change records.
- **Commit:** A cryptographic, permanent snapshot of staged files at a specific point in time, referenced by a unique SHA-1/SHA-256 hash and accompanied by a developer message explaining the change.
- **Branching:** Isolates new feature work, bug fixes, or experimental refactoring into an independent line of development, ensuring the `main` or `production` branch remains clean, deployable, and bug-free.
- **Merge Conflict:** Occurs when two developers modify the exact same line of code in different ways across converging branches. The version control system halts automated merging, requiring engineers to manually review the conflicting lines, choose the correct implementation, and commit the resolved code.

Chapter 3 Practice Questions

Multiple Choice Questions

Question 3.1 [Source: Sample Trial 2025, Q6]

Which of the following is the **least** important consideration when choosing an open-source front-end web framework for a new enterprise online application?

- (A) Popularity of the framework on social media
- (B) Quality of existing framework documentation
- (C) Knowledge of the framework within the company's development team
- (D) Compatibility of the framework with the company's existing technology

Question 3.2 [Source: Claritas Original]

An engineer identifies that a large, uncompressed JavaScript file is positioned within the `<head>` tag of a website, causing significant delays before text and images appear on the screen. Which strategy resolves this issue without moving the script tag?

- (A) Replace the script with an open-source CMS plugin
- (B) Add the `defer` attribute to the `<script>` tag
- (C) Encode the JavaScript file into UTF-8 machine-readable JSON
- (D) Enforce a stricter Content Security Policy header

Short Answer Questions**Question 3.3 (4 marks)** [Source: HSC Familiarisation Questions, Q10]

Consider the following source code for a web page:

```
<html>
  <body>
    <link rel="stylesheet" href="style3.css"/>
    <script src="image-gallery.js"/><!-- this script is slow to load-->
    <link rel="stylesheet" href="styles1.css"/>
    <link rel="stylesheet" href="styles2.css"/>
    <h1>My Awesome Website</h1>
    <p>I hope you like my website, I spent a lot of time designing it.</p>
    <h2>Photos from my holidays</h2>
    
    
    
    
  </body>
</html>
```

Explain how the load time of this page can be improved.

Question 3.4 (4 marks) [Source: Sample Trial 2025, Q17]

Explain why an experienced web developer would choose to use raw HTML and CSS while a beginner would use a Content Management System (CMS).

Chapter 3 Marking Guidelines & Exemplars

Question 3.1 Solution

- **Correct Answer:** (A) Popularity of the framework on social media.
- **Rationale:** Enterprise technology evaluation prioritizes documentation quality, existing team competencies, long-term stability, and architectural compatibility. Social media hype is not a sound technical metric.

Question 3.2 Solution

- **Correct Answer:** (B) Add the `defer` attribute to the `<script>` tag.
- **Rationale:** Adding `defer` allows the browser to download the script in the background while continuing to parse the HTML DOM tree, executing the script only after the document structure is fully built.

Question 3.3 Marking Guide

MARKING CRITERIA	MARKS
• Provides four distinct, technically detailed performance optimisations tailored directly to the provided source code. • Covers image dimensions/formats, script parsing bottlenecks, CSS stylesheet consolidation, and asset placement.	4
• Identifies three practical performance fixes relevant to the code snippet.	3
• Outlines two general load-time improvements.	2
• Identifies one isolated fix (e.g. resize the images).	1

Band 6 Sample Exemplar:

1. **Image Downsampling & Modern Formats:** The HTML forces the browser to download massive 4000×3000 pixel images only to scale them down to 640×480 using HTML attributes. Resizing the source files to exactly 640×480 and converting them to modern compressed formats like WebP or AVIF will reduce file payload sizes by over 90%.
2. **Consolidate Stylesheets:** The page links three separate CSS files (`style3.css` , `styles1.css` , `styles2.css`). Bundling and minifying these into a single CSS file eliminates redundant HTTP network handshakes.
3. **Non-Blocking Script Execution:** The external script `image-gallery.js` is placed high in the document body without asynchronous attributes, blocking the HTML parsing engine. Adding the `defer` attribute (or relocating the tag immediately before `</body>`) allows the HTML content to render without pausing for script downloads.
4. **Implement Lazy Loading:** Adding `loading="lazy"` to the `<img>` tags ensures the holiday photos are only fetched as they approach the user's viewport, speeding up the initial First Contentful Paint.

Question 3.4 Marking Guide

MARKING CRITERIA	MARKS
Clearly explains why experienced developers prefer custom HTML/CSS (fine control, minimal bloat, performance, security) AND why beginners prefer a CMS (low code, pre-built templates, abstracted server operations).	4
Outlines valid reasons for both developer pathways with minor gaps in technical depth.	3
Focuses almost exclusively on one side of the comparison (either experienced or beginner).	2
Provides basic definitions of a CMS and HTML/CSS.	1

Band 6 Sample Exemplar:

An experienced web developer often chooses native HTML and CSS because it provides complete control over the Document Object Model, stylesheet cascades, and critical rendering performance. By writing custom code, they avoid the significant computational bloat, redundant JavaScript libraries, and slow database queries common to generic CMS platforms. Custom development allows engineers to build lightweight, highly responsive architectures that strictly comply with W3C standards, maintain tight security postures, and integrate with modern CI/CD pipelines.

In contrast, a beginner typically selects a Content Management System (such as WordPress) because it abstracts away low-level architectural complexity. A CMS provides an intuitive Graphical User Interface with drag-and-drop page builders, pre-built templates, and modular plugins. This allows non-technical users to design, publish, and maintain dynamic database-driven websites without having to learn syntax, configure server environments, or write manual SQL database queries.

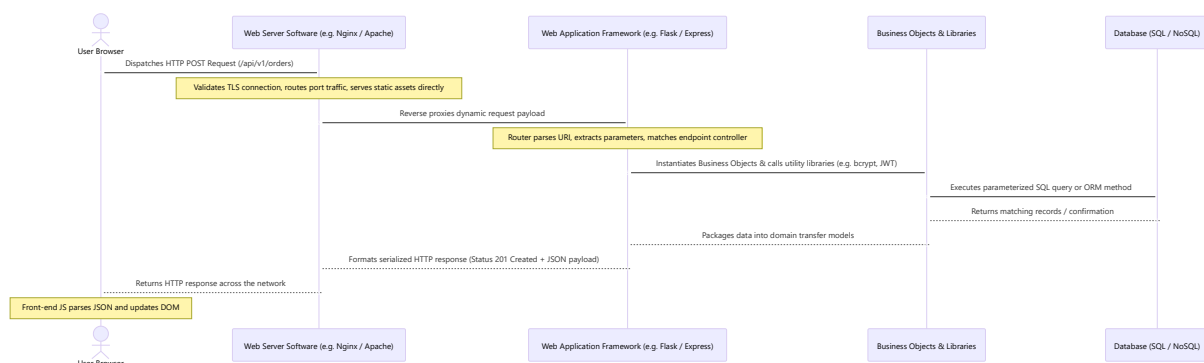
Chapter 4 - Back-End Engineering, Web Requests & Relational Databases

Syllabus Dot Points Covered:

- *Designing web applications* — Model elements that form a web development system: server-side (back-end) web programming, interfacing with databases that are based on Structured Query Language (SQL) or non-SQL
- *Designing web applications* — Assess the contribution of back-end web development to the success of a web application
- *Designing web applications* — Observe and describe the back-end process used to manage a web request: role of webserver software, web framework, objects, libraries, databases
- *Designing web applications* — Develop a web application using an appropriate scripting language with shell scripts to make files and directories, and searching for text in a text file
- *Designing web applications* — Apply a web-based database and construct script that executes SQL: selecting fields, incorporating 'group by', common SQL queries, constraints using WHERE keyword, table joins
- *Designing web applications* — Compare Object-Relational Mapping (ORM) to SQL
- *Designing web applications* — Describe how collaborative work practices between front-end and back-end developers improve the development of a web solution

4.1 The End-to-End Back-End Web Request Lifecycle

While front-end development builds the user-facing interface, the back-end serves as the foundational application engine. It runs on remote servers or cloud environments, handling business logic, authenticating requests, managing data access, and enforcing security boundaries.



Core Server-Side Architecture Components

1. **Web Server Software (e.g., Nginx, Apache):** Sits at the network boundary, listening for incoming TCP traffic on designated ports (80/443). It acts as a reverse proxy, manages SSL/TLS handshakes,

routes static files (images, compiled CSS) directly to reduce latency, and distributes incoming requests across multiple application instances using load-balancing algorithms.

2. **Web Framework (e.g., Python Flask, Django, Node.js Express):** The structured software scaffolding running behind the web server. It provides a centralized **router** that maps incoming HTTP request methods and URI routes (e.g., `GET /products/{id}`) to specific programmatic controller subroutines. It parses incoming payloads, manages cookie sessions, and formats responses.

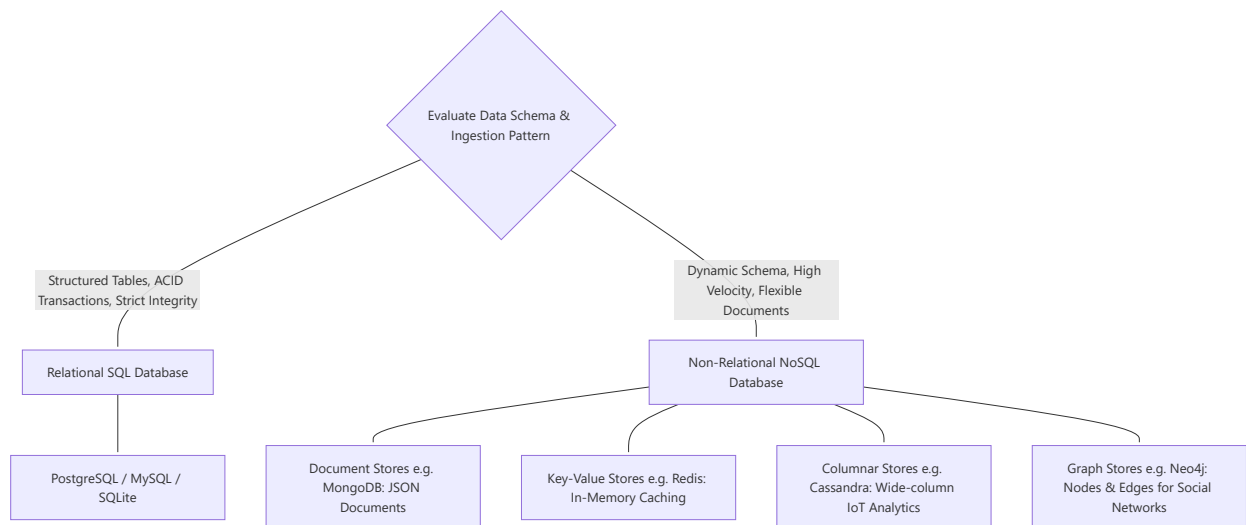
3. **Domain Objects and Libraries:**

- **Objects:** Encapsulate enterprise data structures and state (e.g., a `User`, `Cart`, or `Invoice` class) using Object-Oriented Programming (OOP) paradigms.
- **Libraries:** Reusable, pre-compiled modules imported into the framework to handle complex tasks without reinventing low-level algorithms—such as cryptography libraries (e.g., `bcrypt` for password hashing) and payment gateway SDKs (e.g., Stripe API integrations).

4. **Databases:** Persistent storage engines responsible for storing, modifying, and retrieving application data safely, upholding consistency rules (such as ACID transactions).

4.2 Database Systems: Relational (SQL) vs Non-Relational (NoSQL)

Back-end architectures rely on persistence layers to preserve information. Modern systems select between structured **Relational (SQL)** engines and distributed **Non-Relational (NoSQL)** databases based on application data complexity and throughput demands.



The Four NoSQL Architectural Models

1. **Document-Based Stores (e.g., MongoDB):** Encapsulate related data in flexible, semi-structured JSON or BSON (Binary JSON) documents. Fields can vary across records without requiring universal schema migrations, making them ideal for dynamic catalogues and user profiles.
2. **Key-Value Stores (e.g., Redis):** The fastest persistence mechanism, operating on an associative array model where an arbitrary key maps directly to a discrete value blob. Highly leveraged for session caching, shopping carts, and real-time leaderboards.
3. **Column-Family Stores (e.g., Apache Cassandra):** Organize data by column families rather than two-dimensional row matrices. Built to distribute massive volumes of write-heavy IoT metrics and analytical data over thousands of server clusters.

4. **Graph Databases (e.g., Neo4j):** Built explicitly around mathematical graph theory, modeling entities as **Nodes** and relationships as **Edges**. Designed for highly interconnected networks (e.g., social platform friends, recommendation engines, fraud detection rings).

Architectural Scaling: Horizontal vs Vertical

- **Vertical Scaling (Scale Up):** Increasing the computational power of a single database host by upgrading its physical CPU cores, RAM allocation, or NVMe storage capacity. Relational SQL databases traditionally scale vertically to maintain rigid transaction consistency.
- **Horizontal Scaling (Scale Out):** Distributing database workloads across a cluster of multiple coordinated commodity servers (nodes). NoSQL architectures are engineered natively to scale horizontally, partitioning and replicating shards across cloud instances dynamically.

4.3 Scripting and Shell Automation for Server Operations

Back-end software engineers frequently write shell scripts to automate server management, configure deployment environments, manipulate system files, and inspect operational server logs.

```
#!/bin/bash

# =====
# Automated Log Maintenance and Keyword Extraction Script
# =====

# 1. Create a structured directory hierarchy for storing security archives
TARGET_DIR="./server_logs/security_archives"
mkdir -p "$TARGET_DIR"

# 2. Touch and instantiate a newly indexed audit tracking file
REPORT_FILE="$TARGET_DIR/unauthorized_access_report.txt"
touch "$REPORT_FILE"

# 3. Search for occurrences of security alerts within raw server access logs
# Using grep to locate 401 (Unauthorized) and 403 (Forbidden) HTTP status codes
echo "=== AUDIT EXTRACTION TIMESTAMP: $(date) ===" >> "$REPORT_FILE"
grep -E "HTTP/1\.[01]\\" (401|403)" /var/log/nginx/access.log >> "$REPORT_FILE"

echo "Log extraction complete. Results written to $REPORT_FILE."
```

In HSC programming tasks, you are expected to understand how basic scripting commands interact with operating systems:

- `mkdir -p <path>` : Generates a directory path, automatically creating parent directories if they do not already exist.
- `touch <filename>` : Instantiates an empty file or updates the access timestamp of an existing file.

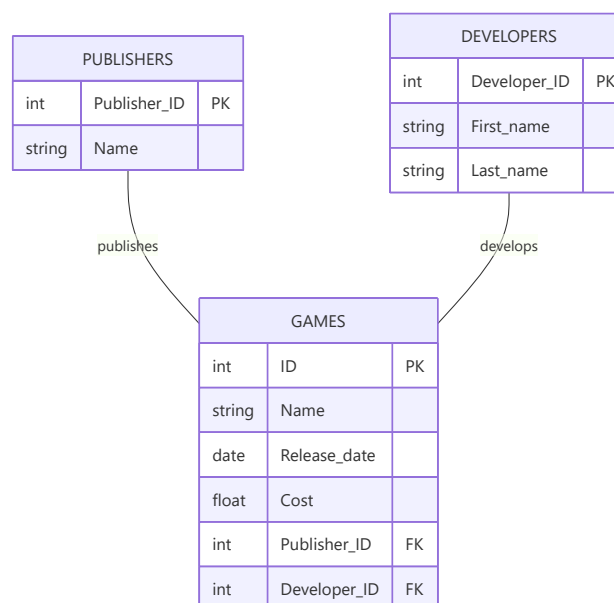
- `grep "<search_string>" <filename>` : Parses the target file line by line, isolating and extracting lines matching the specified text pattern or regular expression.

4.4 Relational Databases & Structured Query Language (SQL)

Relational Database Management Systems (RDBMS)—such as PostgreSQL, MySQL, and SQLite—organize data into structured tables with strict column data types, using primary and foreign keys to model relationships between entities.

Syllabus Definition: Relational Terms

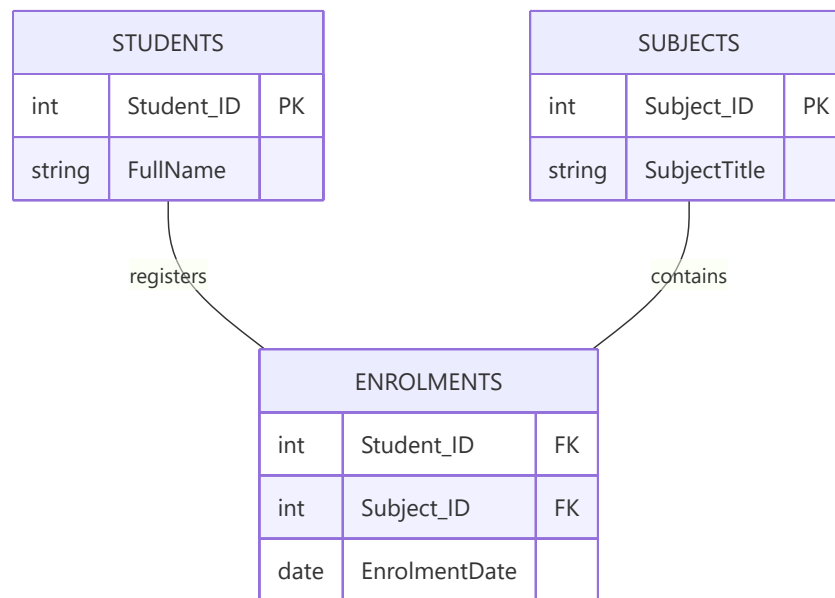
- **Table:** A collection of related data organized into horizontal rows and vertical columns.
- **Field (Column):** A single attribute of an entity, defined by a specific data type and constraints.
- **Record (Row / Tuple):** A single, uniquely identifiable instance of an entity containing values for each field.
- **Primary Key (PK):** A field (or combination of fields) that uniquely identifies each record within a table, preventing duplicate entries.
- **Foreign Key (FK):** A field in one table that references the Primary Key of another table, enforcing **referential integrity** between related datasets.



Database Cardinality and Relationships

Relational architectures model real-world business associations via three primary relationship types:

- **One-to-One (1:1):** A single record in Table A is associated with exactly one record in Table B (e.g., a `Citizen` table linked to a single `Passport` record). Often consolidated into a single entity unless segregated for security partitioning.
- **One-to-Many (1:M):** A single record in Table A relates to multiple records in Table B, but a record in Table B references back to only one in Table A (e.g., one `Customer` can place multiple `Orders`, but each `Order` belongs strictly to one `Customer`). Enforced by placing the Primary Key of Table A as a Foreign Key inside Table B.
- **Many-to-Many (M:N):** Multiple records in Table A link to multiple records in Table B (e.g., `Students` enrolling in multiple `Subjects`, and each `Subject` hosting multiple `Students`). Relational database engines cannot link Many-to-Many entities directly; they require an intermediate **Join Table** (Associative Entity) containing composite Foreign Keys referencing both parents.



Database Normalisation: First Normal Form (1NF)

Normalisation is the systematic database design procedure used to eliminate data redundancy, prevent update anomalies, and enforce structural integrity.

- **First Normal Form (1NF):** A relational table satisfies 1NF if and only if:
 1. Every column attribute contains strictly **atomic** (indivisible) values.
 2. There are no repeating groups or multi-valued arrays within a single record.
 3. Every row is uniquely identifiable via a designated **Primary Key**.

1NF Normalisation Example

Non-1NF Table (Violates Atomicity & Contains Repeating Values):

STUDENTID	STUDENTNAME	ENROLLEDSUBJECTS
101	Sarah Chen	Software Eng, Mathematics, Physics

Normalized into First Normal Form (1NF):

STUDENTID	STUDENTNAME	SUBJECT
101	Sarah Chen	Software Eng
101	Sarah Chen	Mathematics
101	Sarah Chen	Physics

(Each row now represents an atomic, indivisible record uniquely identifiable by the composite primary key `(StudentID, Subject)`).

SQL Syntax Reference (NESA Course Specifications)

Structured Query Language (SQL) is the declarative language used to query and manipulate relational databases. In the HSC Software Engineering examination, queries are evaluated using ANSI SQL syntax adhering to standard clauses:

```
SELECT field1, field2, AGGREGATE_FUNCTION(field3) AS alias_name
FROM TableA, TableB
WHERE TableA.PK_ID = TableB.FK_ID
      AND search_condition
GROUP BY field1, field2
ORDER BY field1 ASC;
```

Essential SQL Clauses & Mechanics

1. **SELECT** : Specifies the fields, calculations, or aggregate expressions to retrieve. The **AS** keyword creates descriptive aliases for output columns.
2. **FROM** : Identifies the participating database tables.
3. **WHERE** : Filters source records before any grouping occurs. Relational operators include `=`, `<>`, `>`, `<`, `>=`, `<=`, `CONTAINS`, `AND`, `OR`, and `NOT`.
4. **Table Joins (Equi-joins)**: Links related tables via their primary and foreign keys, typically defined within the **WHERE** clause (e.g., `WHERE Games.Publisher_ID = Publishers.Publisher_ID`).

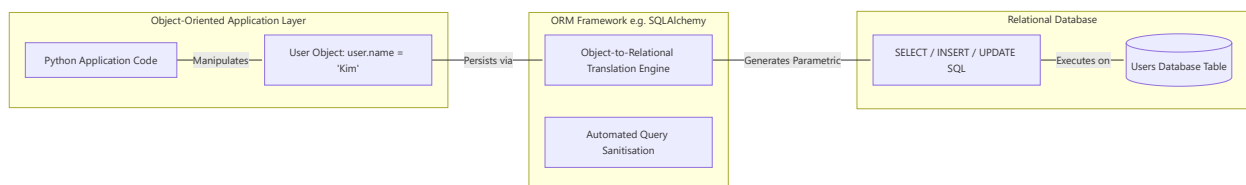
5. **GROUP BY** : Collapses rows sharing identical values across specified attributes into summary rows. It is paired with aggregate functions:
 - **COUNT(attribute)** : Tally of matching non-null rows.
 - **SUM(attribute)** : Total arithmetic sum of a numeric field.
 - **AVG(attribute)** : Arithmetic mean of a numeric field.
 - **MIN(attribute)** / **MAX(attribute)** : Lowest or highest values recorded.
6. **ORDER BY** : Sorts the final result dataset in ascending (**ASC** , default) or descending (**DESC**) order.

4.5 Object-Relational Mapping (ORM) vs Raw SQL

When writing object-oriented back-end code (such as Python, Java, or C#), a conceptual mismatch arises: object-oriented code operates on memory-resident object graphs with inheritance and methods, whereas relational databases operate on flat two-dimensional mathematical tables.

Syllabus Definition

Object-Relational Mapping (ORM): An architectural abstraction layer that bridges object-oriented programming languages and relational databases. An ORM automatically maps database tables to classes, table rows to instantiated objects, and table columns to object attributes, allowing developers to query and manipulate database records using native programming syntax.



Code Comparison: Fetching High-Value Users

- **Raw SQL Implementation (Python SQLite):**

```
# Raw SQL requires explicit connection management and string query construction
cursor.execute("SELECT * FROM Customers WHERE Rating >= ? ORDER BY Username ASC", (90.0,))
results = cursor.fetchall()
for row in results:
    print(f"Customer: {row['Username']}, Rating: {row['Rating']}")
```

- **Object-Relational Mapping Implementation (Python SQLAlchemy-style):**

```
# ORM models tables as classes; rows are handled as native class instances
high_value_customers = Customer.query.filter(Customer.rating >=
90.0).order_by(Customer.username.asc()).all()

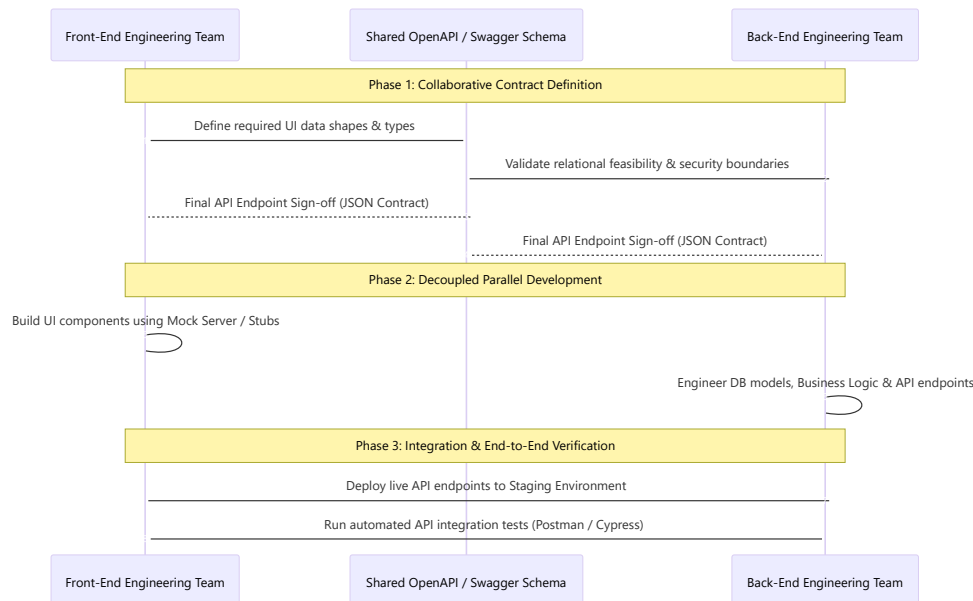
for customer in high_value_customers:
    print(f"Customer: {customer.username}, Rating: {customer.rating}")
```

Detailed Comparison: ORM vs Raw SQL

ENGINEERING METRIC	STRUCTURED QUERY LANGUAGE (SQL)	OBJECT-RELATIONAL MAPPING (ORM)
Level of Abstraction	Low: Engineers write explicit, declarative SQL commands executed directly by the database engine.	High: Database access is abstracted behind native object models and programming language method chains.
Developer Productivity	Moderate: Requires writing boilerplate SQL strings and manually parsing returned array tuples into local variables.	High: Eliminates boilerplate query construction; changes to object properties update the database via built-in methods.
Performance & Optimization	Maximum: Skilled engineers can hand-craft fine-tuned queries, multi-table joins, subqueries, and database indexes for complex operations.	Potential Overhead: In complex queries, ORMs can generate inefficient SQL (e.g., the $N + 1$ query problem), causing performance bottlenecks.
Database Portability	Tied: Raw SQL queries often use database-specific dialects (e.g., T-SQL vs PL/pgSQL); migrating databases requires rewriting queries.	Portable: The ORM abstracts database dialects; switching from SQLite to PostgreSQL usually requires only updating the connection string.
Security (SQL Injection)	Vulnerable if mismanaged: Using raw string concatenation (<code>"WHERE id=" + id</code>) exposes the app to SQL Injection; requires careful parameterization.	Secure by Default: ORMs use parameterized queries and sanitization automatically, mitigating common SQL injection risks.

4.6 Collaborative Practices: Front-End and Back-End Integration

Modern web applications require close collaboration between front-end and back-end development teams. Miscommunications regarding data formats, network routes, or delivery timelines can result in broken deployments and delayed schedules.



Best Practices for Front-End & Back-End Collaboration

- 1. API Contracts First:** Before writing code, both teams collaborate to design the application programming interface (API) using formal specification standards (such as OpenAPI/Swagger). This defines the exact URI paths, required request headers, expected HTTP response codes, and the precise JSON payload schema.
- 2. Parallel Workflows via Stubs and Mock Servers:** Once an API contract is finalized, front-end engineers can develop against a local mock server returning fake data matching the agreed JSON structure. Meanwhile, back-end engineers build the real database models and business logic in parallel, preventing bottlenecks.
- 3. Cross-Functional Communication and Feedback:** Regular daily stand-ups and integration sprints catch edge-case discrepancies early (e.g., handling null values, pagination limits, or slow network response states), avoiding painful last-minute rewrites before production deployment.

Chapter 4 Practice Questions

Multiple Choice Questions

Question 4.1 [Source: 2025 HSC Exam, Q9]

Two approaches that can be used when programming databases for use in web applications are Object-Relational Mapping (ORM) and Structured Query Language (SQL). Match each characteristic to the appropriate approach:

1. Code is focused on object manipulation
2. Tables can be joined to enhance searching
3. Programmers can choose the language used to access the database

Which row represents the correct mapping for Object-Relational Mapping (ORM)?

- (A) 1 and 2 only
 (B) 1 and 3 only
 (C) 2 and 3 only
 (D) 1, 2, and 3

Question 4.2 [Source: Claritas Original]

A back-end server architecture consists of Nginx, a Python Flask application, and a PostgreSQL database. What is the primary role of the Nginx web server software when handling an incoming client web request?

- (A) Translating object-oriented class instances into relational table rows
 (B) Acting as a reverse proxy, managing incoming TCP connections, and routing requests to Flask
 (C) Executing SQL table joins across database indexes
 (D) Manipulating Document Object Model (DOM) elements on the browser

Short Answer Questions

Question 4.3 (4 marks) [Source: 2025 CSSA Trial, Q19b]

A developer has chosen to construct an enterprise web application utilizing Python. Discuss the options of using Object-Relational Mapping (ORM) versus raw Structured Query Language (SQL) to store and manage client data.

Question 4.4 (4 marks) [Source: 2025 HSC Exam, Q21]

Below is the **Issues** table for a technical support company. **FixTime**, in minutes, is the time taken to complete each job.

Table: Issues

JOBID	ISSUE	RESPONSE	FIXTIME	CATEGORY	STATUS
0001	Cannot connect to the internet	Clear cache	22	Software	Pending
0002	Installation failure	Reinstall the printer drivers	76	Hardware	Pending
0004	Forgot password	Reset password	97	Software	Resolved
0005	Installation failure	Follow the installation guide	110	Hardware	Escalated
0009	Blue screen error	Run a system diagnostic test	120	Software	Pending

JOBID	ISSUE	RESPONSE	FIXTIME	CATEGORY	STATUS
0011	Installation failure	Verify email settings	45	Software	Escalated

Construct an SQL query that displays:

- the number of pending software jobs
- the number of pending hardware jobs
- the number of escalated software jobs
- the number of escalated hardware jobs

List the results in order from the highest to lowest based on the number of jobs.

Chapter 4 Marking Guidelines & Exemplars

Question 4.1 Solution

- **Correct Answer:** (B) 1 and 3 only.
- **Rationale:** ORM focuses on manipulating programmatic objects (Characteristic 1) and allows developers to query using their native programming language (Characteristic 3). Joining tables via explicit relational syntax is a foundational feature of raw SQL (Characteristic 2).

Question 4.2 Solution

- **Correct Answer:** (B) Acting as a reverse proxy, managing incoming TCP connections, and routing requests to Flask.
- **Rationale:** Web server software (Nginx/Apache) sits on the front line of server architecture, handling low-level network connections, SSL termination, and reverse-proxying dynamic traffic to the application framework.

Question 4.3 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly discusses both advantages AND disadvantages of ORM AND raw SQL within a Python web development context. • Covers performance implications, abstraction benefits, developer productivity, and query optimization.	4
• Discusses advantages and/or disadvantages of both ORM and SQL with minor omissions.	3

MARKING CRITERIA	MARKS
Describes either ORM or SQL in depth, or provides basic outlines of both.	2
Demonstrates basic familiarity with database querying or ORM terminology.	1

Band 6 Sample Exemplar:

Using raw SQL in Python (e.g., via `sqlite3` or `psycopg2`) gives developers complete, granular control over query design, table joins, and database indexes. This can result in optimized execution times for complex reporting queries. However, writing raw SQL requires engineers to master database-specific dialects, manually parse returned record tuples into Python variables, and maintain queries carefully to prevent SQL Injection vulnerabilities, slowing initial feature development.

Using an Object-Relational Mapping (ORM) framework like SQLAlchemy abstracts database operations behind Pythonic classes and methods, mapping table rows directly to Python objects. This improves developer productivity, allows easier unit testing, and reduces syntax errors. ORMs also provide built-in query parameterization, protecting against SQL injection, and allow migrating across database engines with minimal changes.

However, ORMs introduce computational overhead. For highly complex queries or bulk data operations, the ORM can generate inefficient SQL that increases database latency compared to a hand-crafted query. For standard enterprise applications, an ORM offers the best balance of speed and maintainability, though performance-critical sections may still benefit from raw SQL.

Question 4.4 Marking Guide

MARKING CRITERIA	MARKS
Provides a completely correct SQL query that selects Category and Status, calculates the count using an alias, correctly filters for 'Pending' and 'Escalated' jobs, groups by both required attributes, and sorts descending by count.	4
Provides an SQL query that satisfies most structural requirements (e.g. missing sorting or minor syntax error in filter).	3
Provides an SQL query demonstrating understanding of grouping and aggregation, but contains multiple syntax or logic errors.	2
Identifies basic <code>SELECT</code> and <code>FROM</code> syntax.	1

Band 6 Sample Exemplar:

```
SELECT Category, Status, COUNT(JobID) AS Jobs
FROM Issues
WHERE Status = 'Pending' OR Status = 'Escalated'
GROUP BY Category, Status
ORDER BY Jobs DESC;
```

(Note: `ORDER BY COUNT(JobID) DESC` is also fully accepted under NESA Course Specifications).

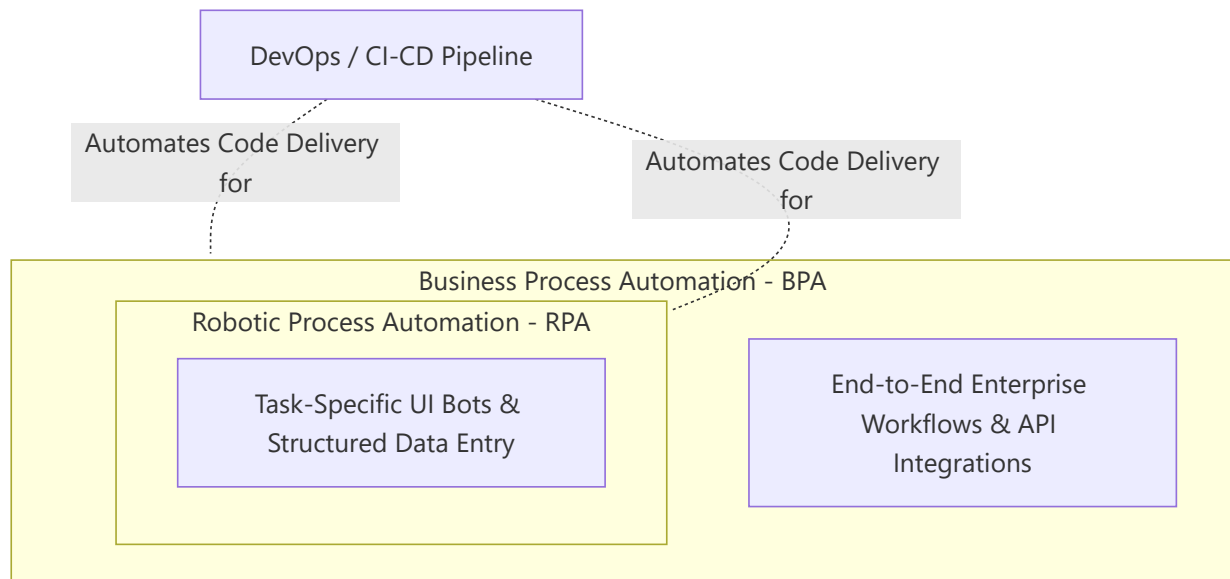
Chapter 5 - Automation Paradigms, DevOps & the MLOps Lifecycle

Syllabus Dot Points Covered:

- *Algorithms in machine learning* — Investigate how machine learning (ML) supports automation through the use of DevOps, robotic process automation (RPA) and business process automation (BPA)
- *Algorithms in machine learning* — Distinguish between artificial intelligence (AI) and ML
- *Course Specifications* — Machine learning automation through DevOps (MLOps): Design, Model Development, and Operations stages

5.1 The Automation Spectrum: RPA, BPA, and DevOps

Software automation replaces manual, repetitive, and error-prone human intervention with programmatic workflows. In enterprise software engineering, automation is categorized across three foundational tiers:



Robotic Process Automation (RPA)

Robotic Process Automation (RPA) deploys software agents ("bots") to mimic rule-based human interactions at the user interface (UI) layer. RPA bots operate on existing graphical interfaces without requiring deep backend database or API modifications.

- **Typical Mechanics:** The bot simulates mouse clicks, keystrokes, data copying between desktop windows, and form submissions.

- **Use Cases:** Extracting tabular data from scanned PDF invoices into an accounting system, parsing employee shift logs from spreadsheets into legacy payroll software, or automatically downloading end-of-day bank balance statements.
- **Limitations:** Fragile against UI changes; if a button's pixel position or DOM ID changes, standard rule-based RPA scripts fail.

Business Process Automation (BPA)

Business Process Automation (BPA) re-engineers and streamlines entire multi-step business workflows across multiple departments and systems, typically integrating at the network and API layers rather than merely mimicking clicks at the UI layer.

- **Typical Mechanics:** Employs event-driven architectures, webhooks, and direct database connectors to route data seamlessly between CRM platforms, inventory databases, payment gateways, and logistics providers.
- **Use Cases:** End-to-end customer onboarding (validating identity → running automated credit checks → generating contracts → creating database accounts → provisioning service access → sending confirmation notifications).
- **RPA vs BPA:** RPA automates discrete, repetitive **tasks** within an existing interface; BPA coordinates multi-system **workflows** across an entire organization.

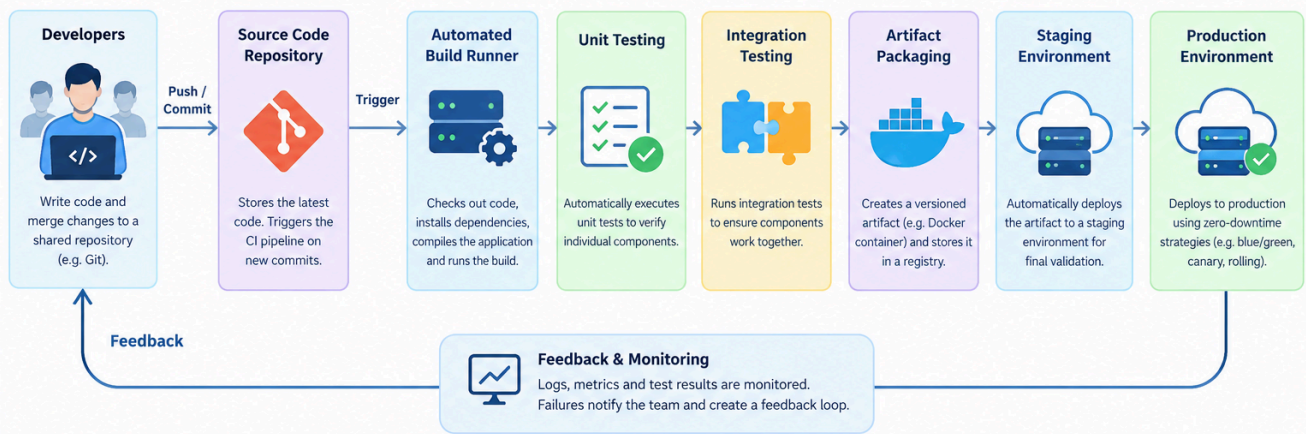
DevOps (Development and Operations)

DevOps is not an automation tool itself, but rather an organizational engineering philosophy supported by automated practices. It unifies software development (Dev) and IT operations (Ops) to shorten the systems development life cycle and deliver high-quality software continuously.

- **Continuous Integration (CI):** Developers merge code into a shared repository frequently. Automated build tools immediately compile the application and execute unit test suites, providing rapid feedback.
- **Continuous Delivery / Deployment (CD):** Validated code artifacts are automatically packaged, configured, and deployed to staging or production cloud environments without manual intervention.

DevOps – CI/CD Pipeline

Unifying Development and Operations to deliver high-quality software continuously.



Key Benefits



Faster delivery

Shortens the systems development life cycle.



Higher quality

Automated testing catches issues early.



Greater reliability

Consistent, repeatable deployments.

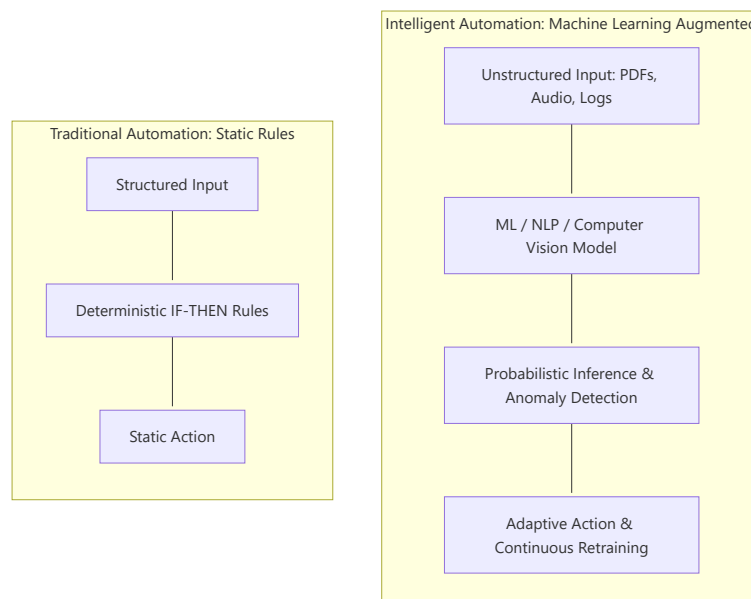


Better collaboration

Unifies Dev and Ops teams around a shared goal.

5.2 Enhancing Automation with Machine Learning (Intelligent Automation)

Traditional RPA, BPA, and DevOps rely on deterministic, static "if-this-then-that" rules. When applied to ambiguous real-world data, rule-based systems break down. **Machine Learning (ML)** introduces statistical intelligence, pattern recognition, and predictive capabilities, transforming static automation into **Hyperautomation (Intelligent Automation)**.



ML Supporting DevOps (AIOps)

- **Log Telemetry Anomaly Detection:** Machine learning algorithms parse gigabytes of real-time server telemetry and application logs. Unlike threshold alerts, ML identifies subtle deviations from normal baseline behavior, predicting server failures or memory leaks before outages occur.
- **Intelligent Test Prioritisation:** ML models analyze Git commit histories and modified code modules, predicting which test cases in a massive testing suite are most likely to fail, executing them first to accelerate CI/CD feedback cycles.
- **Predictive Resource Auto-Scaling:** Regression models predict upcoming web traffic spikes based on time of day, seasonal factors, and marketing campaigns, spinning up cloud containers proactively rather than reactively.

ML Supporting RPA (Intelligent Process Automation - IPA)

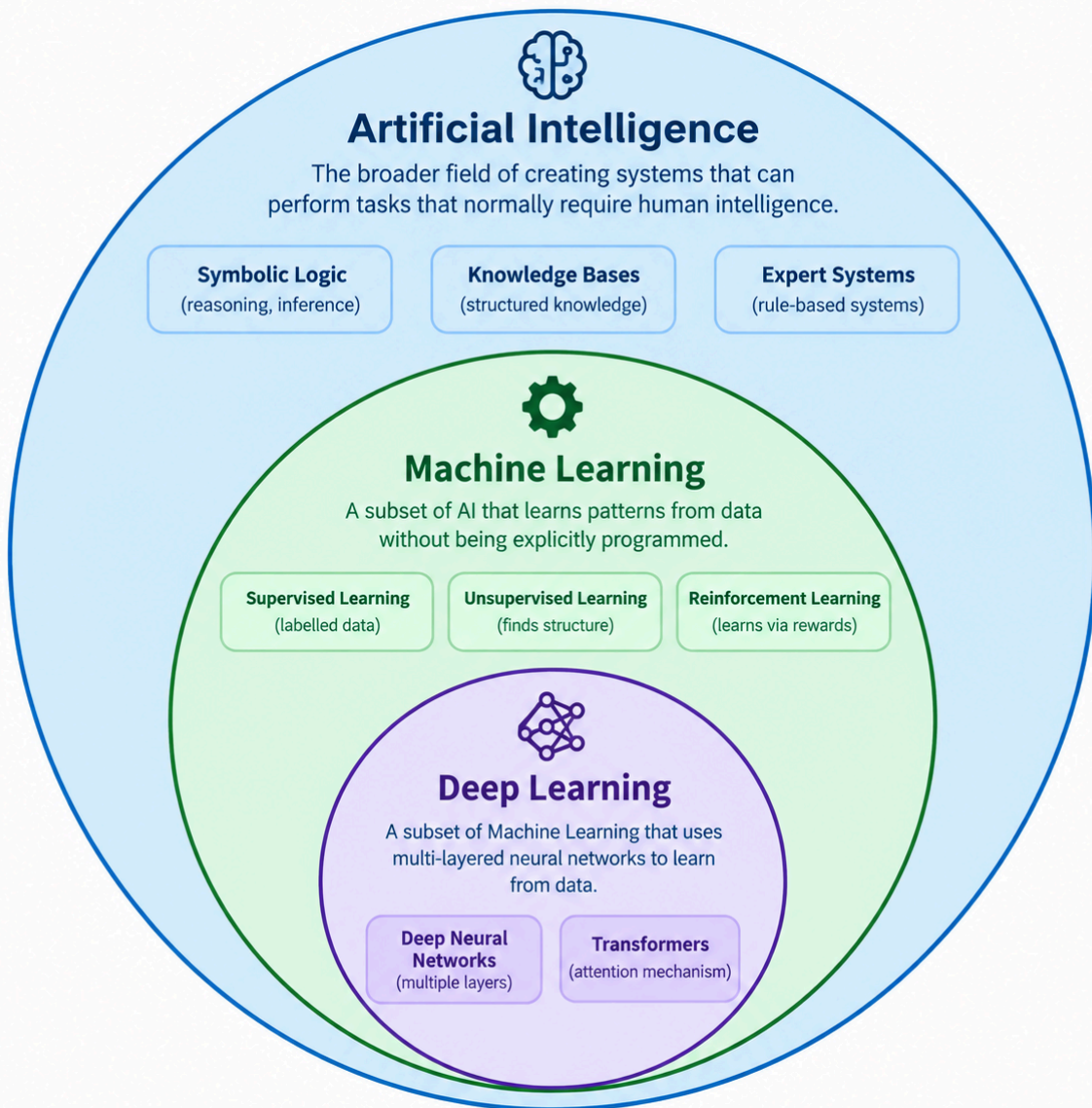
- **Processing Unstructured Data:** Optical Character Recognition (OCR) combined with Natural Language Processing (NLP) enables RPA bots to extract meaning from non-standard documents. A bot can parse unstructured customer complaint emails or varied invoice layouts, extracting account numbers and dollar amounts regardless of document format.
- **Cognitive Decisioning:** Instead of halting when an edge case occurs, the RPA bot routes the transaction to an embedded classification model that calculates a confidence score. High-confidence decisions proceed automatically; ambiguous transactions are flagged for human review.

ML Supporting BPA

- **Process Mining & Bottleneck Discovery:** Unsupervised clustering algorithms inspect enterprise workflow logs to discover where approvals stall or identify redundant steps across corporate pipelines.
- **Dynamic Workflow Routing:** Rather than routing loan applications via rigid salary thresholds, predictive models assess complex multidimensional risk profiles, dynamically routing applicants to automated approval or manual underwriting.

5.3 Defining the AI and ML Taxonomy

The terms Artificial Intelligence and Machine Learning are frequently conflated in commercial marketing, yet they represent distinct concepts in computer science.



Syllabus Definition: AI vs ML

- **Artificial Intelligence (AI):** The broad discipline of engineering machines and software systems capable of simulating human intelligence, including reasoning, general problem-solving, environmental perception, and planning. AI encompasses both deterministic, rule-based expert systems and data-driven learning models.
- **Machine Learning (ML):** A specific subset of Artificial Intelligence focused on developing algorithms that iteratively learn from data and improve their task performance through experience, rather than following rigid, pre-programmed, hand-coded rules.

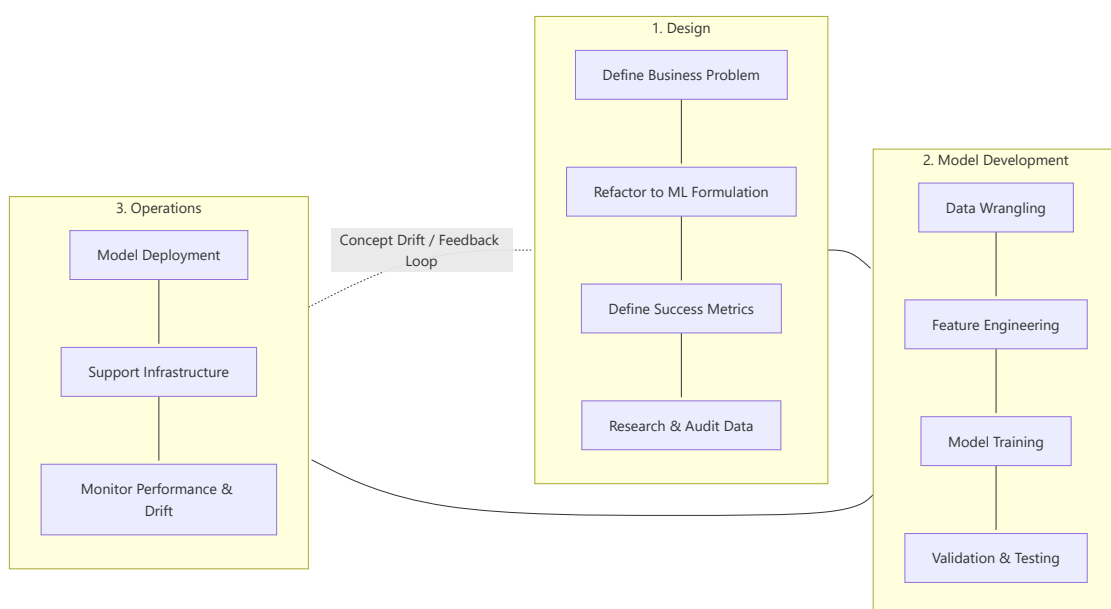
DIMENSION	ARTIFICIAL INTELLIGENCE (AI)	MACHINE LEARNING (ML)
Scope	Broad superset encompassing all intelligent machines.	Specific subset and methodology within AI.

DIMENSION	ARTIFICIAL INTELLIGENCE (AI)	MACHINE LEARNING (ML)
Core Mechanism	Rule engines, formal logic, knowledge graphs, search trees, and statistical models.	Statistical algorithms, parameter optimization, loss functions, and weight adjustments.
Data Dependency	Variable; expert systems can run entirely on deterministic, hard-coded rules without training data.	Absolute; cannot function without structured or unstructured training datasets.
Objective	Simulate human-like cognitive behavior across specific or generalized domains.	Extract patterns from historical data to optimize predictions or classify unseen inputs.

5.4 The Machine Learning Operations (MLOps) Lifecycle

Developing machine learning software requires more than training a model locally in a script. Models degrade as real-world data drifts over time. **Machine Learning Operations (MLOps)** adapts DevOps principles to the machine learning lifecycle, establishing automated pipelines for developing, deploying, and maintaining production models.

According to the **NESA Software Engineering Course Specifications**, MLOps comprises three interconnected, iterative stages:



Stage 1: Design

1. **Defining the Business Problem:** Clearly articulating the operational objective (e.g., "The e-commerce platform loses \$2M annually due to customers abandoning carts").
2. **Refactoring into an ML Problem:** Translating the business challenge into a concrete machine learning formulation (e.g., converting cart abandonment into a binary classification problem: predicting the probability $P(\text{abandon} = 1 \mid X)$).
3. **Defining Success Metrics:** Establishing measurable technical and business thresholds (e.g., achieving an F1-score > 0.85 and inference latency $< 50\text{ms}$).
4. **Researching Available Data:** Auditing data sources for availability, historical depth, sample volume, and compliance with privacy regulations.

Stage 2: Model Development

1. **Data Wrangling:** Cleaning raw data by handling missing values, imputing nulls, removing duplicates, and eliminating noisy outliers.
2. **Feature Engineering:** Transforming raw attributes into meaningful mathematical representations (e.g., scaling numerical variables, one-hot encoding categories, computing session duration from raw timestamps).
3. **Model Training:** Fitting candidate algorithms (e.g., Linear Regression, Decision Trees, Neural Networks) to training data by iteratively optimizing internal parameters and loss functions.
4. **Model Testing and Validation:** Evaluating generalizability on holdout test datasets to verify that the model has not overfitted to noise.

Stage 3: Operations

1. **Model Deployment:** Packaging the trained model artifact into a containerized microservice and exposing an API endpoint (e.g., via a Flask or FastAPI REST service).
2. **Supporting Operations and Infrastructure:** Allocating compute resources (CPU/GPU), configuring load balancers, and managing concurrency.
3. **Monitoring Model Performance:** Continuously tracking real-time inference accuracy, latency, and **data drift** (when incoming production data diverges from the historical training distribution, requiring automated model retraining).

Chapter 5 Practice Questions

Multiple Choice Questions

Question 5.1 [Source: 2025 HSC Exam, Q11]

Which of the following are key principles of DevOps? Select all that apply.

- ☐ Rapid feedback
- ☐ Freedom from bias
- ☐ Procedural fairness

- ☐ Automation of repetitive tasks
- ☐ Collaboration and shared ownership

Question 5.2 [Source: 2025 CSSA Trial, Q6]

Which of the following best distinguishes artificial intelligence (AI) from machine learning (ML)?

- (A) AI is a subset of ML.
- (B) ML requires human input, while AI does not.
- (C) ML focuses on hardware, while AI focuses on software.
- (D) AI focuses on building systems that simulate human intelligence, while ML focuses on enabling machines to learn from data.

Short Answer Questions

Question 5.3 (3 marks) [Source: APHS 2025 Trial, Q7]

Social Media Platform X tracked the duration of time 16–24 year old males spent watching video categories (Gaming, News, Entertainment) to optimize content suggestions. Explain how process automation could have been used to generate and log this data without human intervention.

Question 5.4 (4 marks) [Source: 2025 CSSA Trial, Q23]

BJcomputing Solutions Inc. has developed a machine learning product recommendation model with high test accuracy, but faces severe difficulties deploying and maintaining it as customer trends change. Analyse how the engineering team should apply the three stages of Machine Learning Operations (MLOps) to solve this challenge.

Chapter 5 Marking Guidelines & Exemplars

Question 5.1 Solution

- **Correct Options:**
 - ☒ Rapid feedback
 - ☒ Automation of repetitive tasks
 - ☒ Collaboration and shared ownership
- **Rationale:** DevOps emphasizes continuous delivery pipelines that automate repetitive build/test cycles, provide fast feedback to developers via CI/CD, and dismantle organizational silos through shared operational ownership. Freedom from bias and procedural fairness are ethical principles of data governance, not DevOps tenets.

Question 5.2 Solution

- **Correct Answer:** (D) AI focuses on building systems that simulate human intelligence, while ML focuses on enabling machines to learn from data.
- **Rationale:** AI is the broad umbrella discipline covering all machines exhibiting intelligent behavior, whereas ML is the specific statistical sub-discipline enabling systems to infer patterns directly from data.

Question 5.3 Marking Guide

MARKING CRITERIA	MARKS
- Explains how process automation (RPA or BPA) automatically collects, tracks, and logs user interaction data (e.g. video watch time, categories) continuously in the background without human intervention. - Identifies the interaction between client telemetry and backend storage.	3
Describes how automated tracking collects user activity data with minor omissions.	2
States a basic feature of automation or data collection.	1

Band 6 Sample Exemplar:

Robotic Process Automation and backend tracking scripts monitor user client-side interactions in the background without requiring manual data entry. Telemetry scripts continuously track video playback events, timestamps, pauses, fast-forwards, and category tags at millisecond precision.

As users scroll through feeds, automated Business Process Automation pipelines ingest these event streams via an API gateway, associate the duration data with demographic user IDs (e.g. 16–24 male cohorts), and automatically write structured log records to central analytics databases, providing a clean historical dataset for training recommendation algorithms.

Question 5.4 Marking Guide

MARKING CRITERIA	MARKS
- Thoroughly explains how all THREE stages of MLOps (Design, Model Development, and Operations) are applied to the deployment and maintenance challenges described in the scenario. - Explicitly connects operational monitoring to feedback loops and automated retraining.	4
Explains two MLOps stages in detail or outlines all three stages with minor omissions.	3
Outlines one or two stages of MLOps.	2
Demonstrates basic familiarity with machine learning development or deployment.	1

Band 6 Sample Exemplar:

To bridge the gap between their accurate prototype and continuous production reliability, the team should execute the three MLOps stages:

1. **Design:** The team must refactor the e-commerce recommendation problem into explicit ML objectives, define technical and business success metrics (such as click-through rates and sub-100ms API response times), and establish automated data pipelines sourcing real-time browsing telemetry.
 2. **Model Development:** Engineers must construct automated data-wrangling and feature-engineering pipelines to clean incoming user data and train recommendation models iteratively, validating performance against offline test sets to avoid overfitting.
 3. **Operations:** The team must containerize the model into a microservice deployed behind an API gateway with automated CI/CD testing. Crucially, operations must include telemetry dashboards monitoring model latency and **data drift**. When customer seasonal trends shift, the operational monitoring pipeline automatically triggers retraining on fresh data, ensuring recommendations remain accurate without manual redeployments.
-

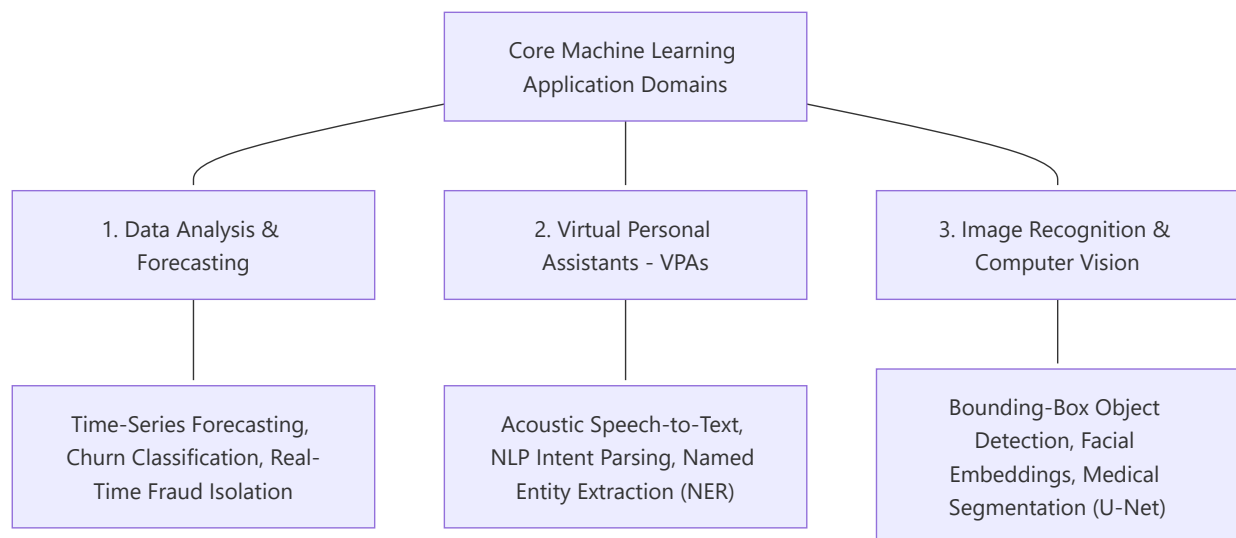
Chapter 6 - Machine Learning Algorithms & Training Models

Syllabus Dot Points Covered:

- *Algorithms in machine learning* — Explore models of training ML: supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning
- *Algorithms in machine learning* — Describe types of algorithms associated with ML: linear regression, logistic regression, K-nearest neighbour
- *Algorithms in machine learning* — Research models used by software engineers to design and analyse ML: decision trees
- *Course Specifications* — Regression algorithms: linear, polynomial, and logistic regression applications

6.0 Common Real-World Applications of Key ML Algorithms

Machine learning models power ubiquitous commercial technologies. Rather than operating in isolation, practical systems combine specialized algorithms to process multidimensional data across three dominant domains:



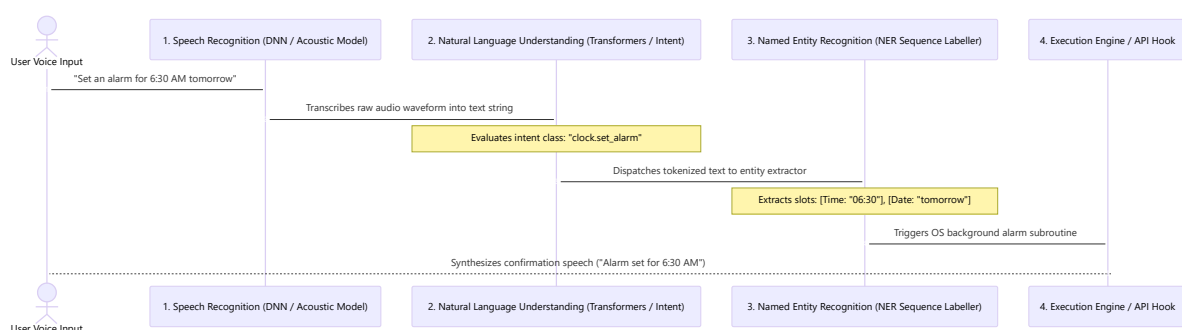
1. Data Analysis and Forecasting

- **Forecasting Financial & Market Trends:** Sequential algorithms (such as Recurrent Neural Networks [RNNs] and Long Short-Term Memory [LSTM] networks) ingest historical pricing time series, trading volumes, and macroeconomic indicators to capture long-term temporal dependencies and forecast market movements.

- **Customer Churn Prediction:** Telecommunications and SaaS providers deploy classification models (Logistic Regression, Random Forests, Gradient Boosted Trees) on subscriber metrics (account tenure, ticket logs, data usage dips). The model calculates individual churn probabilities, triggering automated retention workflows before contract cancellation occurs.
- **Demand and Inventory Forecasting:** Retail logistics deploy non-linear regression models to predict localized stock depletion based on seasonal shifts, weather anomalies, and promotion calendars, preventing supply stockouts while minimizing warehousing overhead.
- **Real-Time Fraud Detection:** Banking systems process transactions through anomaly detection models (e.g., Isolation Forests) and supervised classifiers. By analyzing geographical velocity (e.g., card transactions occurring in Sydney and London within thirty minutes), atypical transaction amounts, and merchant categories, transactions are flagged or rejected in milliseconds.

2. Virtual Personal Assistants (VPAs)

Voice-activated agents (such as Apple Siri, Google Assistant, and Amazon Alexa) integrate an end-to-end pipeline of specialized ML models:



- **Automated Speech Recognition (ASR):** Deep Neural Networks process analog audio waveforms converted into spectrograms, matching acoustic sound frequencies to the most probable phonemes and generating plain-text transcriptions.
- **Intent Recognition:** Natural Language Understanding (NLU) models (such as Transformer networks) parse the transcribed string, classifying the underlying user goal (e.g., mapping *"Remind me to call Mum at five"* to the intent `reminder.create`).
- **Named Entity Recognition (NER):** Token-classification models identify and extract semantic variables (dates, durations, geo-locations, contact names) necessary to populate API execution parameters.
- **Personalized Recommendation Systems:** Machine learning models utilize **collaborative filtering** (identifying users with similar tastes) and **content-based filtering** (matching metadata profiles) to recommend media, music playlists, and predictive calendar actions.

3. Image Recognition & Computer Vision

Computer vision software enables machines to extract spatial and visual meaning from digital image arrays and video streams:

Computer Vision Tasks – Autonomous Driving Example

The same scene, different outputs. From a single image, computer vision can classify, detect and understand what's in the scene.

1 Image Classification

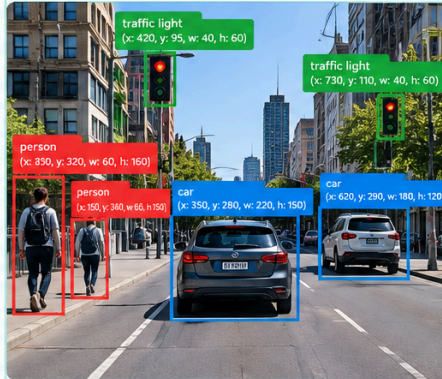
Predicts the overall category of the image.



Predicted Label:
Street scene

2 Object Detection

Finds and labels individual objects with bounding boxes.



Detected Objects (with bounding box coordinates):

Car	(x: 350, y: 280, w: 220, h: 150)
Car	(x: 620, y: 290, w: 180, h: 120)
Person	(x: 80, y: 320, w: 60, h: 160)
Person	(x: 150, y: 340, w: 60, h: 150)
Traffic light	(x: 420, y: 95, w: 40, h: 60)
Traffic light	(x: 730, y: 110, w: 40, h: 60)

3 Semantic Segmentation

Assigns a class to every pixel in the image.



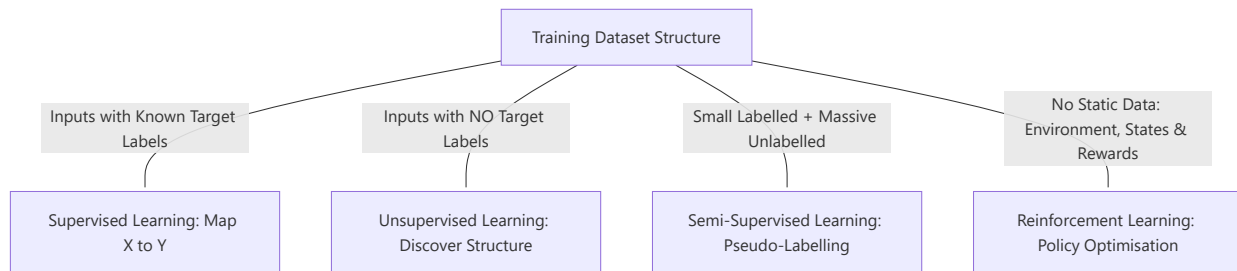
Segmentation Classes (coloured mask):

Road	Building
Sidewalk	Tree
Car	Traffic light
Pedestrian	Sky

- **Object Detection (Autonomous Systems):** Deep Convolutional Neural Networks (CNNs) simultaneously predict object class probabilities and generate exact **bounding box coordinates** around multiple entities (pedestrians, cyclist lanes, stop signs) in real-time video streams.
- **Facial Recognition & Vector Embeddings:** The neural network transforms face pixels into a compact numerical vector known as a **feature embedding**. Identity verification is achieved by calculating the mathematical Euclidean distance between the new facial embedding vector and an encrypted reference database record.
- **Medical Image Segmentation:** Architectures like U-Net classify every individual pixel in magnetic resonance imaging (MRI) or CT scans, delineating healthy organ tissue from malignant tumors to assist oncologists with surgical planning.
- **Optical Character Recognition (OCR):** Hybrid CNN-RNN networks scan handwritten or photographed text, segmenting character glyphs and mapping pixel contours into machine-readable ASCII or Unicode strings.

6.1 The Four Core Machine Learning Training Paradigms

Machine learning models learn patterns through different training paradigms, defined by the presence, absence, or structure of supervisory feedback in the dataset.



1. Supervised Learning

The model is provided with a training dataset consisting of input features paired with ground-truth output labels:

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

The algorithm iteratively adjusts its internal parameters to minimize the difference (loss) between its predictions ($\hat{y}$) and the actual target values (y).

- **Classification:** Predicts a discrete categorical class label (e.g., identifying whether an incoming email is "Spam" or "Not Spam", classifying handwritten digits 0--9).
- **Regression:** Predicts a continuous numerical value (e.g., forecasting house prices, temperature, or vehicle speed).

2. Unsupervised Learning

The model receives input features with **no output labels** ($\mathcal{D} = \{x_1, x_2, \dots, x_n\}$). The algorithm explores the inherent geometric, statistical, or mathematical structures within the data without human guidance.

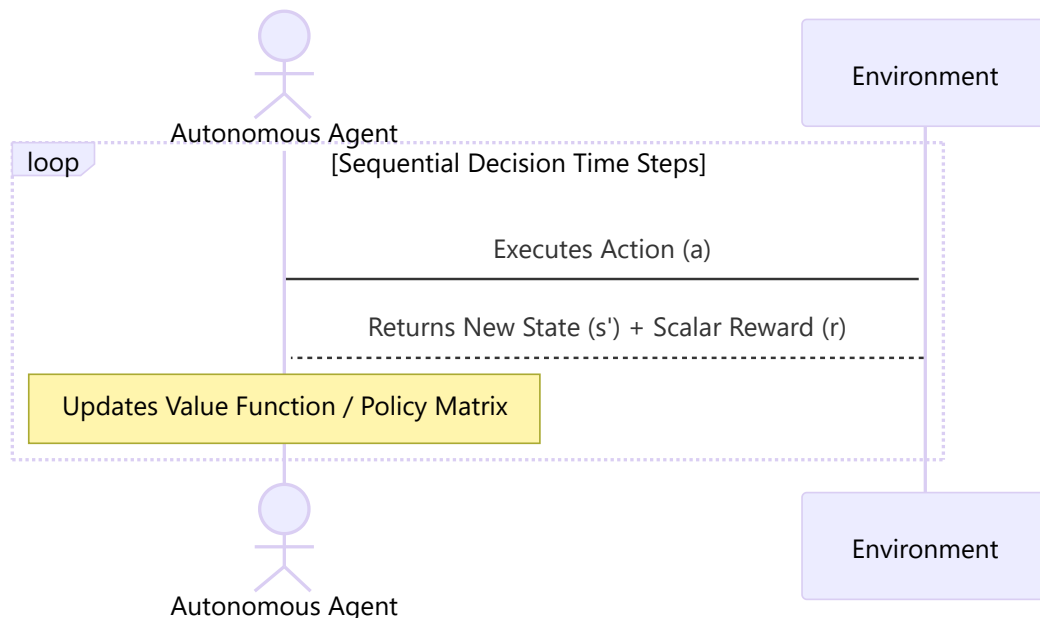
- **Clustering:** Groups similar data points together based on distance metrics (e.g., segmenting customer demographics using K-Means clustering).
- **Dimensionality Reduction:** Compresses hundreds of feature dimensions into fewer variables while retaining critical variance (e.g., Principal Component Analysis [PCA]).
- **Anomaly Detection:** Identifies rare data instances that diverge significantly from the general population distribution (e.g., credit card fraud detection).

3. Semi-Supervised Learning

Addresses situations where unlabelled data is plentiful (e.g., millions of medical scan images), but acquiring verified labels requires expensive human expert labor. The model trains on a small set of verified labelled examples to generate preliminary predictions on the unlabelled pool (**pseudo-labelling**), iteratively incorporating the most confident examples back into its training set.

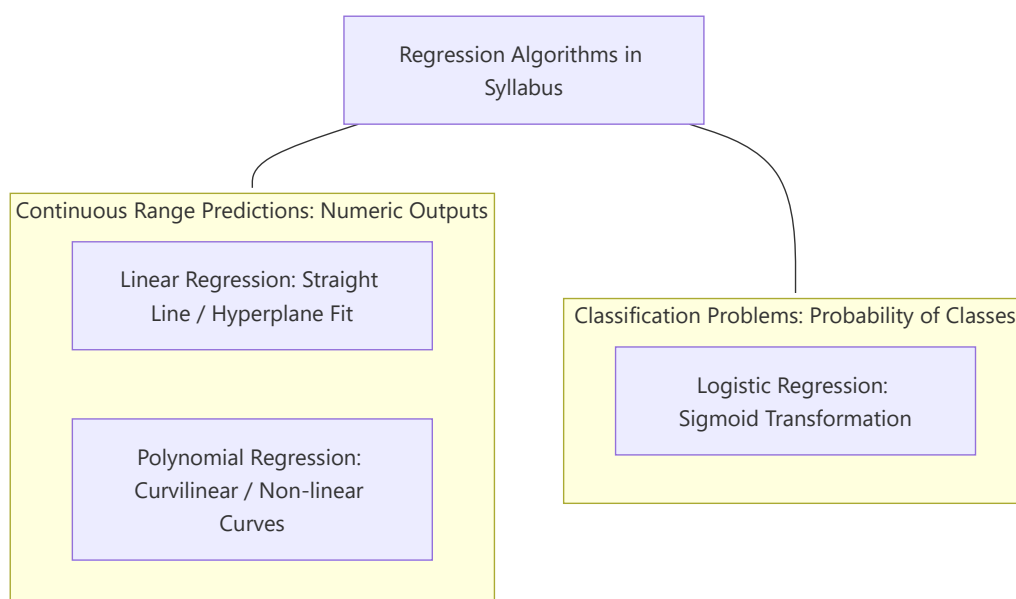
4. Reinforcement Learning (RL)

Rather than ingesting static datasets, an autonomous **agent** learns through trial and error by interacting with a dynamic **environment**. At each time step, the agent observes the current **state** (s), executes an **action** (a), transitions to a new state (s'), and receives scalar feedback in the form of a **reward** (r) or penalty. The agent's goal is to learn an optimal **policy** (π) that maximizes cumulative rewards over time.



6.2 Regression Algorithms: Linear, Polynomial, and Logistic

Regression algorithms establish mathematical relationships between input variables (features) and target variables.



Linear Regression

Assumes a straight-line (linear) relationship between a single independent variable (x) and a dependent continuous variable (y):

$$y = mx + b \quad \text{or} \quad \hat{y} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

The algorithm uses Ordinary Least Squares (OLS) to find the line of best fit that minimizes the sum of

squared vertical residuals:

$$\text{Loss} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

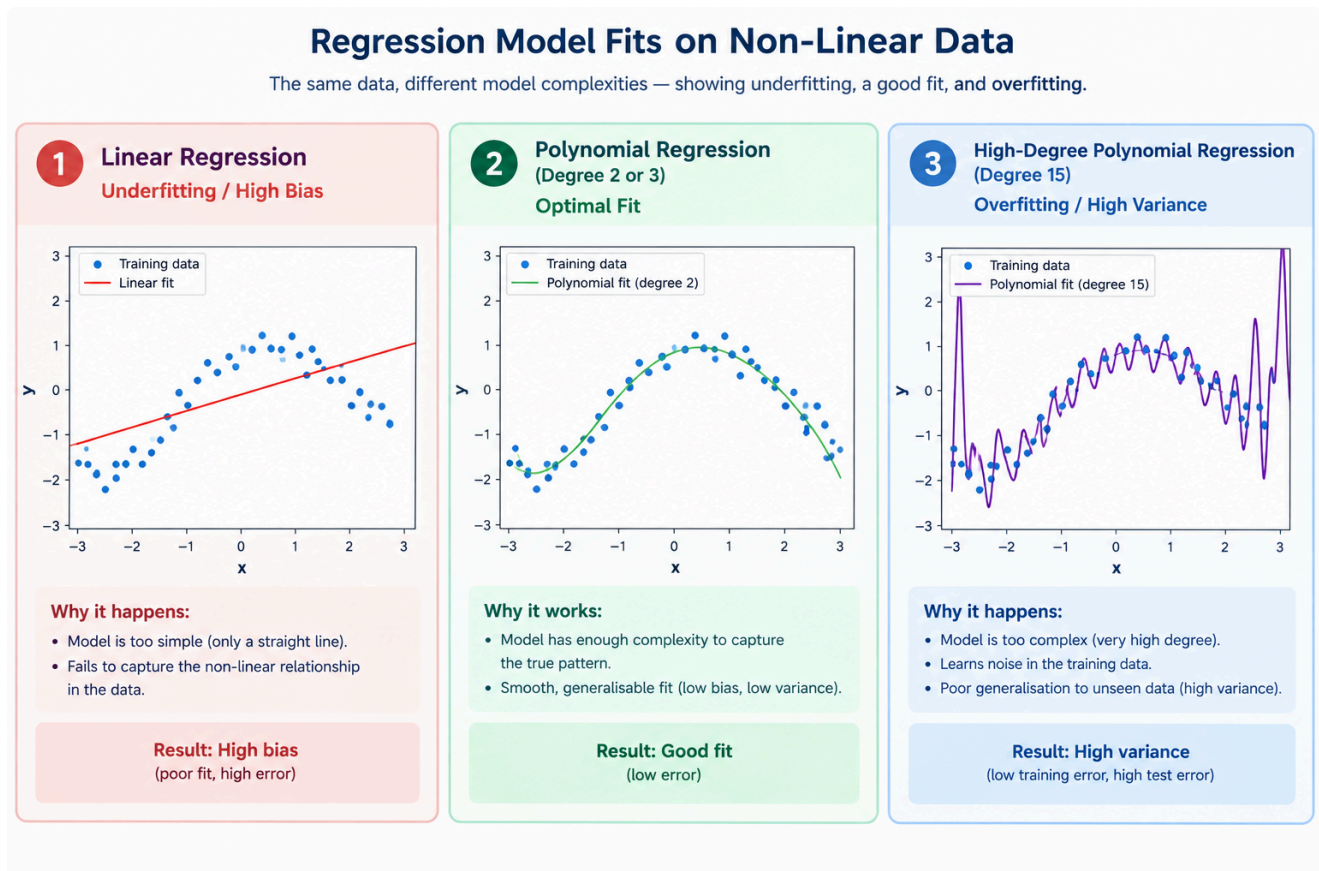
- **Best For:** Simple relationships where changes in x cause proportional changes in y (e.g., predicting factory power consumption based purely on operating hours).
- **Limitations:** Prone to underfitting when real-world data curves or experiences diminishing returns; highly sensitive to extreme outliers.

Polynomial Regression

When data exhibits curvilinear patterns, linear models fail. Polynomial regression fits a non-linear curve by introducing higher-degree powers of the input feature:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \dots + \beta_d x^d$$

- **Best For:** Cyclical, seasonal, or accelerating physical and financial systems (e.g., modeling projectile trajectories, seasonal retail spikes, or engine efficiency across RPM bands).
- **Limitations:** Choosing too high a degree ($d > 4$) causes severe **overfitting**, where the curve oscillates wildly to hit every training data point, failing on new data.

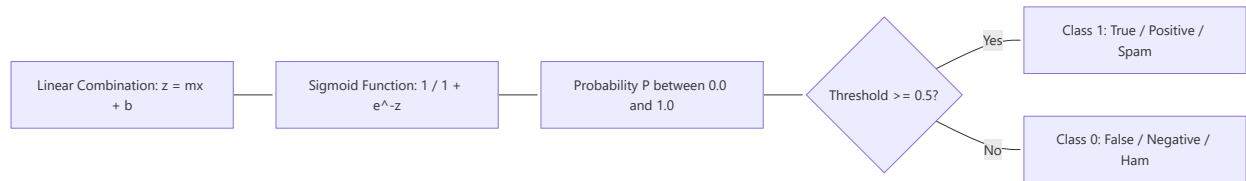


Logistic Regression

Despite its name, **logistic regression is used for classification problems**, not continuous numeric prediction. It predicts the probability that a given input belongs to a discrete binary class (e.g., $y \in \{0, 1\}$).

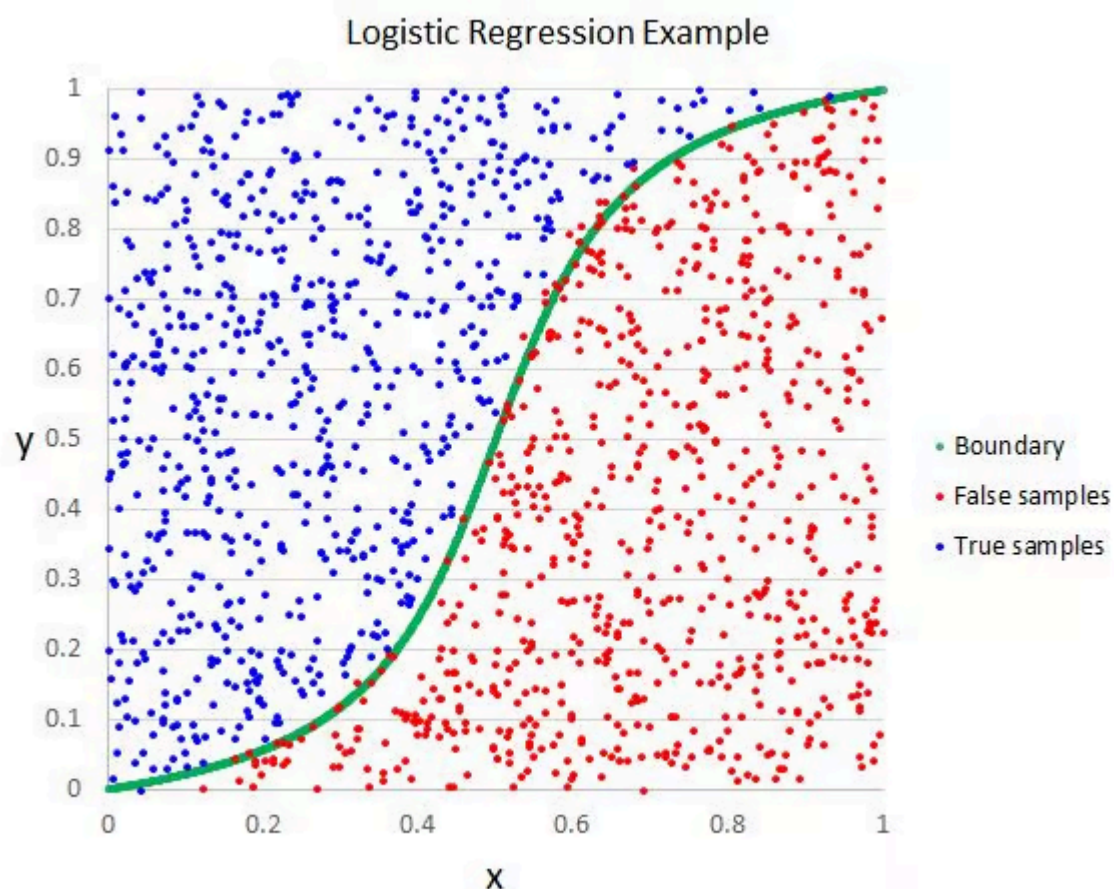
Logistic regression passes a linear combination of inputs through the non-linear **Sigmoid (logistic) function**:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{where} \quad z = \beta_0 + \beta_1 x$$



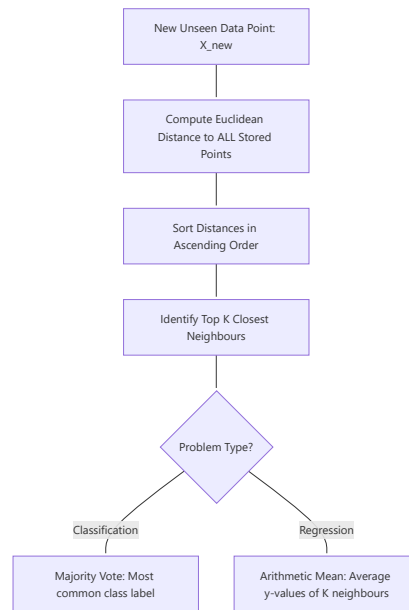
The output $\sigma(z)$ is constrained strictly between 0.0 and 1.0. A classification threshold (typically 0.5) converts this continuous probability into a discrete prediction.

- **Use Cases:** Determining whether an email is spam (1) or legitimate (0), predicting whether a student passes (1) or fails (0), or whether a medical scan indicates disease presence.



6.3 K-Nearest Neighbours (KNN)

K-Nearest Neighbours (KNN) is a non-parametric, lazy-learning algorithm. It does not learn an explicit mathematical equation during training; instead, it stores the entire training dataset in memory.



The Distance Metric

For an input vector with m features, the geometric separation between the new query point (x) and an existing training point (p) is calculated using the **Euclidean Distance formula**:

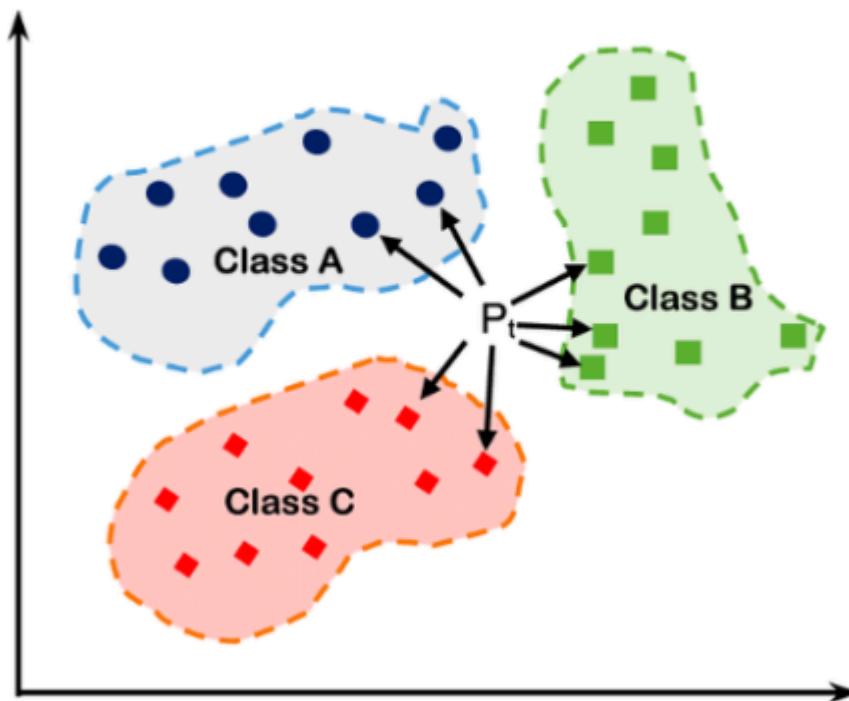
$$d(x, p) = \sqrt{\sum_{j=1}^m (x_j - p_j)^2}$$

Regression Mechanics vs Classification Mechanics

- **KNN Classification:** The algorithm counts the class labels of the K closest neighbors and assigns the majority class to the new data point.
- **KNN Regression:** The algorithm identifies the K closest neighbors and calculates the arithmetic mean of their continuous target values:

$$\hat{y} = \frac{1}{K} \sum_{i=1}^K y_i$$

K Nearest Neighbors

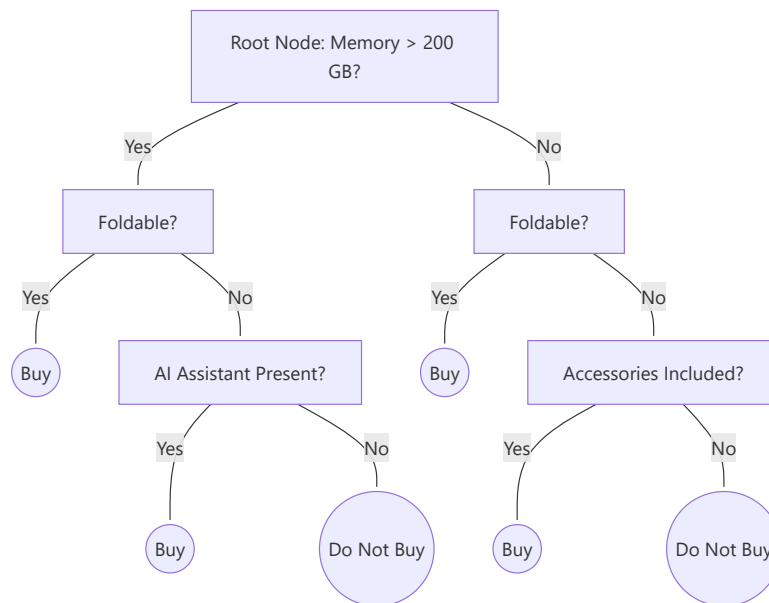


⚠️ HSC Trap: The Critical Role of Hyperparameter K

If K is set too small ($K = 1$), the model is overly sensitive to noise and outliers (overfitting). If K is set too large (e.g., $K = n$), the model predicts the global average of the dataset, ignoring local patterns (underfitting). In binary classification, K is typically chosen as an odd integer (e.g., $K = 3, 5$) to prevent voting ties.

6.4 Decision Trees: Construction & Simplification

A **Decision Tree** is a hierarchical, supervised learning model that splits data recursively based on feature thresholds to reach a classification or regression outcome.



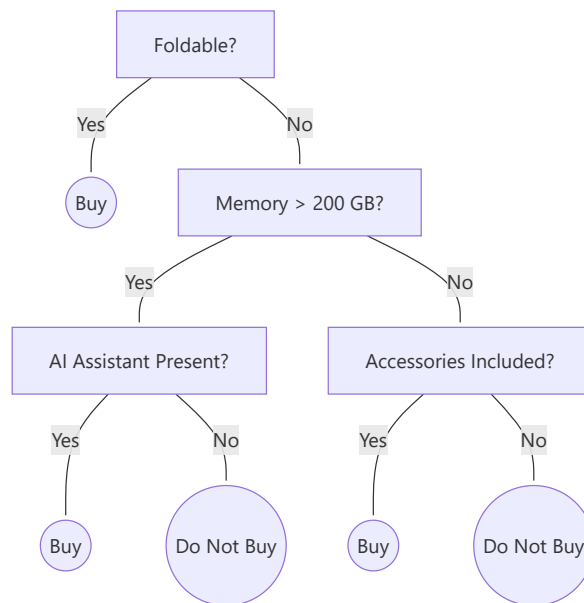
Decision Tree Anatomy

- **Root Node:** The initial node representing the entire dataset, evaluated on the feature providing the highest information gain.
- **Internal (Decision) Nodes:** Intermediate checkpoints testing a specific condition (e.g., $\text{Age} \leq 30$ or $\text{Type} = \text{'SUV'}$).
- **Branches:** Directed paths connecting nodes based on the binary or categorical evaluation outcome of the condition.
- **Leaf (Terminal) Nodes:** The final endpoints providing the prediction; leaf nodes contain no child branches.

Tree Simplification (Pruning)

Complex decision trees with redundant branches waste computational memory and lead to overfitting. Software engineers simplify trees by identifying dominant features that yield identical outcomes regardless of secondary conditions.

In the mobile phone purchasing tree above, notice that whenever $\text{Foldable} = \text{Yes}$, the outcome is **always "Buy"**, regardless of whether memory is above or below 200 GB. Promoting `Foldable` to the root node eliminates duplicate subtrees:



6.5 Model Performance Evaluation Metrics

Evaluating machine learning models requires different statistical metrics depending on whether the task is classification or regression.

Classification Evaluation: The Confusion Matrix

For a binary classification model (e.g., predicting whether a transaction is Fraudulent [Positive] or Legitimate [Negative]):

	ACTUAL POSITIVE ($Y=1$)	ACTUAL NEGATIVE ($Y=0$)
Predicted Positive ($\hat{y}=1$)	True Positive (TP)	False Positive (FP) (Type I Error)
Predicted Negative ($\hat{y}=0$)	False Negative (FN) (Type II Error)	True Negative (TN)

- **Accuracy:** The overall proportion of correct classifications:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Accuracy can be misleading when evaluating imbalanced datasets (e.g., if 99% of transactions are legitimate, a model predicting "Legitimate" for every input achieves 99% accuracy while catching zero fraud).

- **Precision:** The proportion of predicted positive cases that were truly positive (measures reliability):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall (Sensitivity):** The proportion of actual positive cases that the model detected (measures coverage):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1-Score:** The harmonic mean of precision and recall, balancing both metrics when classes are imbalanced:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Regression Evaluation Metrics

- **Mean Squared Error (MSE):** Averages the squared differences between actual and predicted values, heavily penalizing large outliers:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- **Mean Absolute Error (MAE):** Averages the absolute magnitude of errors, providing an interpretable error metric in the original units of measurement:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Chapter 6 Practice Questions

Multiple Choice Questions

Question 6.1 [Source: 2025 CSSA Trial, Q2]

Fill in the blank with the correct term:

____ Regression is used for classification problems (e.g., spam detection).

- (A) Linear
- (B) Polynomial
- (C) Logistic
- (D) K-Nearest

Question 6.2 [Source: 2025 HSC Exam, Q12]

For which of the following situations would logistic regression be used when training a machine learning model to make predictions? Select all that apply.

- ☐ Identifying whether a received email is spam
- ☐ Determining the outcome of a two-party election
- ☐ Calculating house prices based on size of the land
- ☐ Investigating the relationship between height and weight
- ☐ Estimating the rate at which a contagious illness will spread

☐ Working out if someone will be admitted to a university course

Short Answer Questions

Question 6.3 (3 marks) [Source: 2025 HSC Exam, Q23b]

Compare linear regression and the K-nearest neighbour (KNN) algorithm.

Question 6.4 (4 marks) [Source: Sample Trial 2025, Q20]

Describe the K-Nearest Neighbour (KNN) algorithm for regression, predicting the output value $\hat{y}$ for a new data point using a training dataset of (x, y) values given a parameter value of $K = 3$.

Chapter 6 Marking Guidelines & Exemplars

Question 6.1 Solution

- **Correct Answer:** (C) Logistic.
- *Rationale:* Logistic regression applies the sigmoid function to calculate probabilities for categorical classification tasks. Linear and polynomial regressions predict continuous numeric values.

Question 6.2 Solution

- **Correct Options:**
 - ☒ Identifying whether a received email is spam (*Binary: Spam / Not Spam*)
 - ☒ Determining the outcome of a two-party election (*Binary: Candidate A / Candidate B*)
 - ☐ Calculating house prices based on size of the land (*Continuous numeric → Linear/Polynomial*)
 - ☐ Investigating the relationship between height and weight (*Continuous correlation → Linear*)
 - ☐ Estimating the rate at which a contagious illness will spread (*Continuous rate → Non-linear regression*)
 - ☒ Working out if someone will be admitted to a university course (*Binary: Admitted / Rejected*)

Question 6.3 Marking Guide

MARKING CRITERIA	MARKS
• Comprehensively compares linear regression and KNN across model assumptions, computational trade-offs, and boundary shapes. • Identifies that linear regression is parametric while KNN is non-parametric.	3
• Outlines key differences between linear regression and KNN with minor omissions.	2
• States an isolated feature of linear regression or KNN.	1

Band 6 Sample Exemplar:

Linear regression is a parametric algorithm that assumes a direct linear relationship between independent features and a continuous target variable, fitting a single global line of best fit ($y = mx + b$). Once trained, its parameters are fixed, making real-time inference fast using minimal memory.

In contrast, K-Nearest Neighbours (KNN) is a non-parametric, instance-based algorithm that makes no assumptions about data distribution. KNN stores all training data in memory and predicts values based on the local similarity of the K closest data points in geometric space. While KNN can capture complex, highly irregular non-linear boundaries that linear regression cannot model, it is much more computationally intensive during inference because it must calculate distances to every stored point for each new query.

Question 6.4 Marking Guide

MARKING CRITERIA	MARKS
- Provides a precise, sequential description of KNN regression for $K = 3$. - Explicitly details: (1) distance computation, (2) sorting/selecting 3 nearest points, and (3) calculating the arithmetic mean of their y-values.	4
Outlines the main steps of KNN regression but omits distance formulas or confuses regression with classification voting.	3
Describes basic nearest-neighbour concepts with partial accuracy.	2
Mentions finding nearest points without mathematical detail.	1

Band 6 Sample Exemplar:

To predict the continuous target value $\hat{y}_{\text{new}}$ for an unseen query point x_{new} using KNN regression with $K = 3$:

- Distance Calculation:** The algorithm computes the mathematical distance (typically Euclidean distance: $d = \sqrt{(x_{\text{new}} - x_i)^2}$) between the new point and every existing training point (x_i, y_i) in the dataset.
- Neighbor Selection:** The calculated distances are sorted in ascending order. The algorithm identifies the three training instances ($K = 3$) with the smallest distance values to x_{new} .
- Aggregation via Averaging:** Unlike KNN classification (which tallies a majority vote of categorical labels), KNN *regression* computes the arithmetic mean of the actual continuous target values (y_1, y_2, y_3) of these three closest neighbors:

$$\hat{y}_{\text{new}} = \frac{y_1 + y_2 + y_3}{3}$$

4. **Output Generation:** This computed average is returned as the final predicted numeric value for the query point.
-

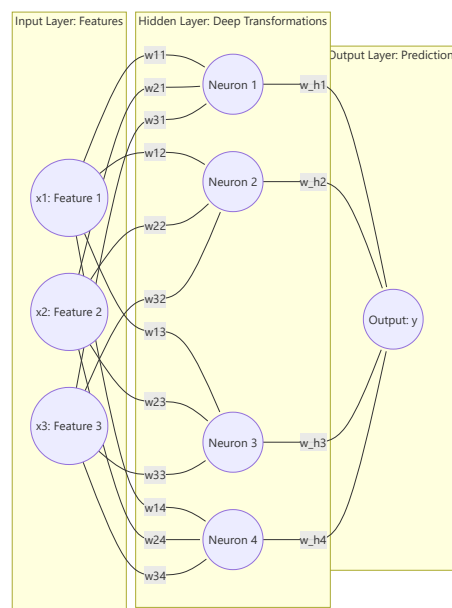
Chapter 7 - Neural Networks & Object-Oriented ML Architecture

Syllabus Dot Points Covered:

- *Algorithms in machine learning* — Research models used by software engineers to design and analyse ML: neural networks
- *Algorithms in machine learning* — Investigate common applications of key ML algorithms: virtual personal assistants, image recognition
- *Programming for automation* — Design, develop and apply ML regression models using an OOP to predict numeric values
- *Programming for automation* — Apply neural network models using an OOP to make predictions
- *Course Specifications* — Neural networks: training cycle (weights, threshold values, linear regression with backward chaining) vs execution cycle

7.1 Neural Network Architecture: Layers, Neurons, and Connections

Artificial Neural Networks (ANNs) are computational models inspired by biological neural processing in animal brains. They are structured into interconnected layers of simple processing nodes called **artificial neurons**.



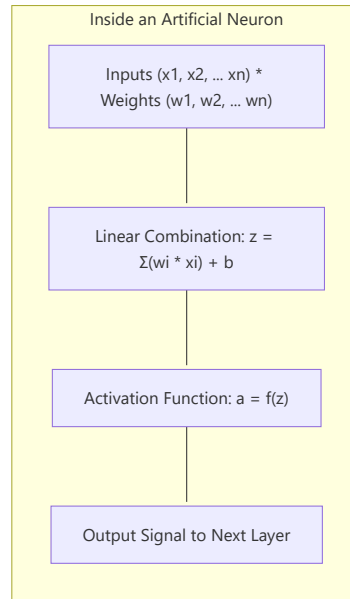
Structural Layers of an ANN

1. **Input Layer:** Receives raw numerical features directly from the dataset. It performs no mathematical transformations; each neuron corresponds to one input variable.

2. **Hidden Layers:** One or more intermediate layers between the input and output. These layers extract non-linear feature representations from the incoming signals.
3. **Output Layer:** Produces the final network prediction. For binary classification, it contains a single neuron generating a probability; for multi-class classification, it contains one neuron per category (typically normalized via a Softmax function); for continuous regression, it outputs an unconstrained real number.

The Mathematical Mechanics of an Artificial Neuron

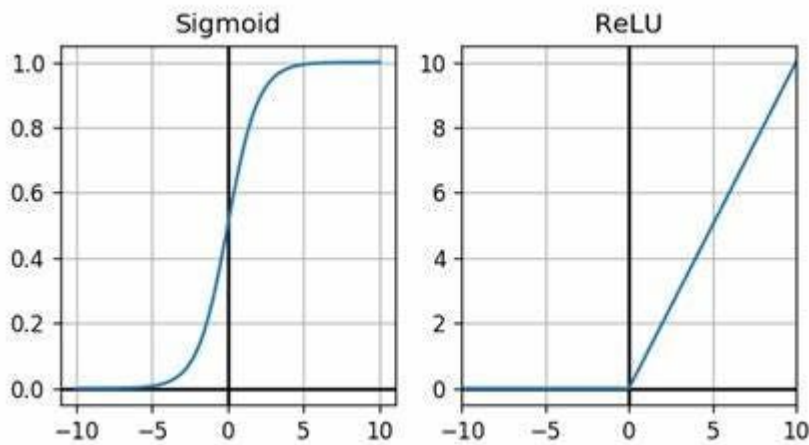
Each neuron processes incoming signals via two distinct operations:



1. **Weighted Summation:** The neuron calculates a linear combination of all incoming inputs (x_i) multiplied by their respective connection weights (w_i), adding an internal **bias** (b):

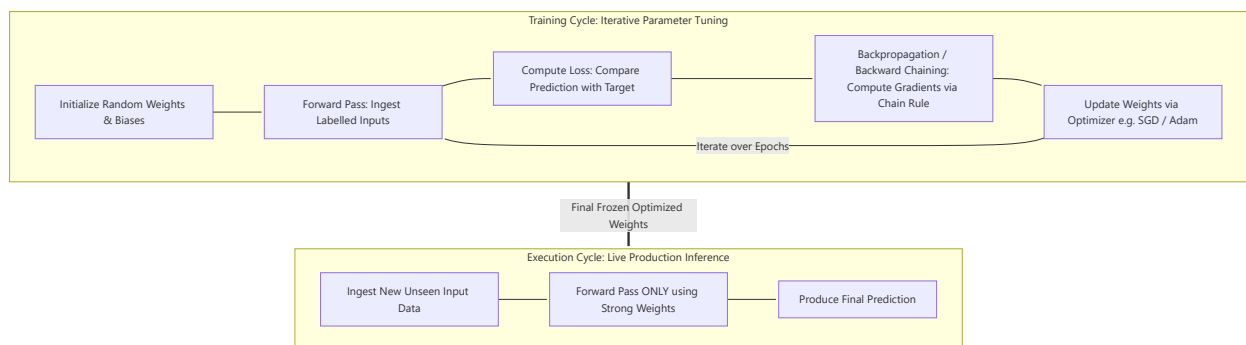
$$z = \sum_{i=1}^m (w_i x_i) + b = (w_1 x_1 + w_2 x_2 + \dots + w_m x_m) + b$$

2. **Activation Function:** The scalar sum z is passed through a non-linear mathematical activation function $f(z)$. Without activation functions, a multi-layer neural network collapses into a basic linear model, incapable of learning complex patterns.
 - **ReLU (Rectified Linear Unit):** $f(z) = \max(0, z)$. Computationally efficient; widely used in intermediate hidden layers.
 - **Sigmoid:** $f(z) = \frac{1}{1+e^{-z}}$. Maps outputs between 0 and 1; ideal for binary output layers.



7.2 The Dual Operational Cycles: Training vs Execution

According to the [NESA Software Engineering Course Specifications](#), understanding neural networks requires distinguishing between the **Training Cycle** and the **Execution (Inference) Cycle**.



The Training Cycle

During training, the network is presented with inputs alongside their known ground-truth outputs.

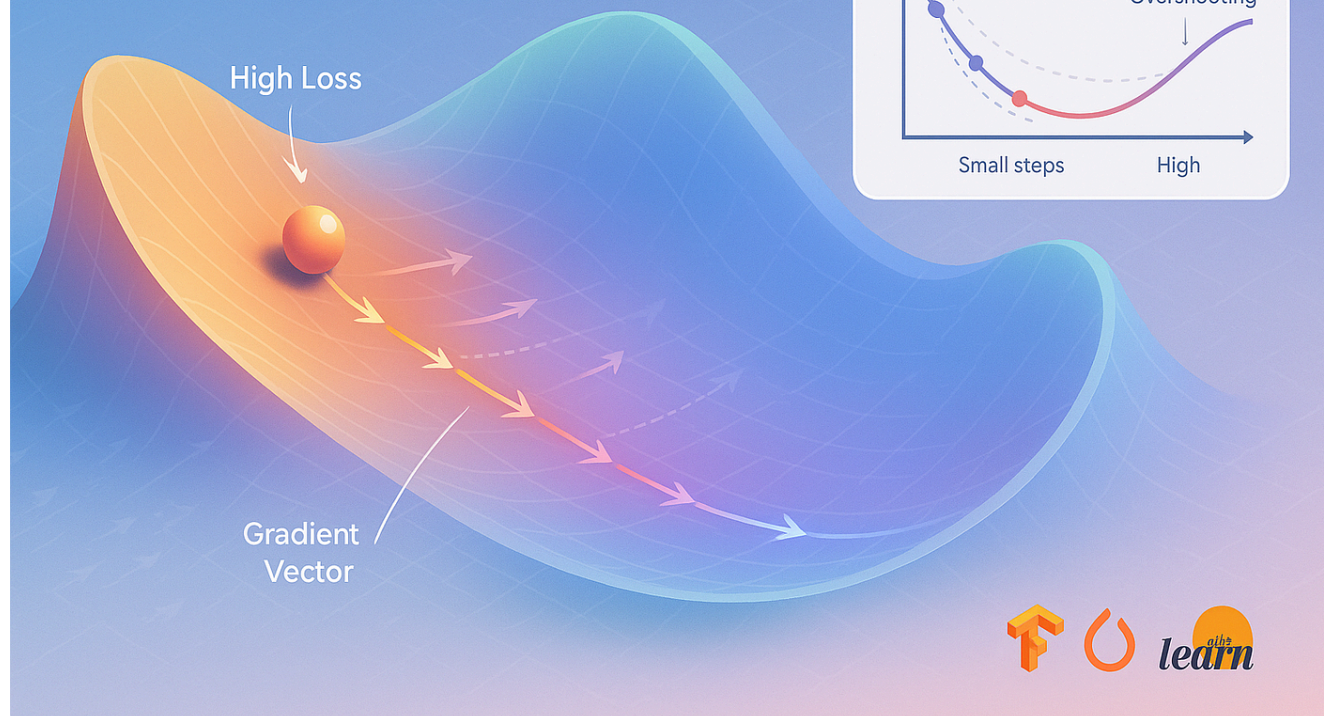
1. **Forward Pass:** Signals propagate from input to output, generating an initial prediction ($\hat{y}$).
2. **Loss Evaluation:** A loss function measures the discrepancy between $\hat{y}$ and actual target y .
3. **Backpropagation (Backward Chaining):** The loss is propagated backward through the network layers. Using calculus (the mathematical chain rule), the system calculates the partial derivative of the error with respect to each internal weight ($\frac{\partial L}{\partial w}$).
4. **Weight Adjustments:** An optimization algorithm (such as Gradient Descent) shifts the connection weights in the direction that reduces the error:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

(where η represents the learning rate). This cycle repeats over many training epochs until the network reaches high pattern-matching accuracy.

Gradient Descent:

How Models Learn by Rolling Downhill



The Execution Cycle

In the execution (inference) cycle, the network is deployed to production with its internal connection weights and biases **frozen**.

- New, unlabelled data is fed into the input layer.
- The data flows exclusively forward through the network.
- Signals travel across connections, where connections with the largest learned weights have the strongest influence on the final output.
- The execution cycle is computationally lightweight compared to training, as it involves only basic matrix multiplications without gradient calculations or backpropagation.

7.3 Object-Oriented Machine Learning Architecture

Modern machine learning systems rely on Object-Oriented Programming (OOP) paradigms to package data, parameters, and algorithms into clean, reusable abstractions.

Claritas Exam Tip: How OOP Supports ML Automation

In HSC questions asking how OOP enhances ML pipelines, structure your answer around:

1. **Encapsulation:** Model hyperparameters (learning rate, weights), data scalers, and training routines are bundled inside a self-contained class, preventing global state pollution.
2. **Abstraction:** A standardized interface (e.g., `.fit()`, `.predict()`, `.evaluate()`) hides complex mathematical algorithms behind simple method calls.
3. **Polymorphism & Inheritance:** Different algorithms (Linear Regression, Neural Networks) can inherit from a base `Estimator` class, allowing automated CI/CD pipelines to swap and benchmark different models without rewriting integration code.

Standard OOP Implementation: NumPy and Scikit-Learn (Course Specifications)

The following implementation shows how machine learning models are constructed using the standard Python Scikit-learn OOP paradigm specified in the NESAs Course Specifications:

```
# =====
# Linear Regression Model using NumPy and Scikit-Learn OOP Framework
# NESAs Course Specifications Reference Pattern
# =====

import numpy as np
from sklearn.linear_model import LinearRegression

# 1. Prepare structured feature matrix (x) and continuous target array (y)
# Features must be a 2-dimensional array/matrix where each row is an instance
x = np.array([[2], [4], [6], [8], [10], [12], [14], [16]])
y = np.array([1, 3, 5, 7, 9, 11, 13, 15])

# 2. Instantiate the model class (OOP Object Construction)
model = LinearRegression()

# 3. Fit the model (Training Cycle: Ordinary Least Squares optimization)
model.fit(x, y)

# 4. Use model object to predict unknown values (Execution / Inference Cycle)
# Predicting for an unseen input value of 4.5
query_feature = np.array([[4.5]])
y_prediction = model.predict(query_feature)

# Display predicted output (Expect 3.5 given the linear relationship y = x - 1)
print(f"Predicted target value for x=4.5: {y_prediction[0]:.2f}")
```

Object-Oriented Polynomial Regression Implementation

When a single linear line is insufficient to model curved trends (such as projectile arcs, pricing elasticity, or biological growth), engineers use polynomial regression. The following implementation uses the standard Scikit-learn Object-Oriented paradigm:

```
# =====  
# Polynomial Regression Model using Scikit-Learn OOP Framework  
# Predicts continuous non-linear numerical values  
# =====  
  
import numpy as np  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression  
  
# 1. Feature data: Machine operating speed (RPM in thousands)  
# Target data: Fuel consumption rate (Litres / hour) showing curved acceleration  
x = np.array([[1.0], [2.0], [3.0], [4.0], [5.0], [6.0]])  
y = np.array([5.2, 7.8, 13.1, 21.5, 34.0, 51.2])  
  
# 2. Instantiate OOP Transformer to generate quadratic polynomial terms (x, x^2)  
poly_transformer = PolynomialFeatures(degree=2)  
x_polynomial = poly_transformer.fit_transform(x)  
  
# 3. Instantiate and train the underlying Linear Regression estimator object  
poly_model = LinearRegression()  
poly_model.fit(x_polynomial, y)  
  
# 4. Predict continuous fuel consumption for an unseen engine speed (4.5 RPM)  
x_query = np.array([[4.5]])  
x_query_poly = poly_transformer.transform(x_query)  
y_prediction = poly_model.predict(x_query_poly)  
  
print(f"Predicted Fuel Consumption for 4.5 RPM: {y_prediction[0]:.2f} L/h")
```

Object-Oriented Logistic Regression Implementation

Unlike linear regression, **logistic regression is used for classification problems**. It maps arbitrary inputs into a probability bounded between 0.0 and 1.0 via the sigmoid function:

```
# =====
# Logistic Regression Model using Scikit-Learn OOP Framework
# Predicts discrete class membership (0 = Ham, 1 = Spam)
# =====

import numpy as np
from sklearn.linear_model import LogisticRegression

# 1. Features: [Word count of 'FREE', Number of external hyperlinks]
# Target: 0 = Legitimate Email (Ham), 1 = Malicious Email (Spam)
x_train = np.array([
    [0, 1], # Legitimate: No 'FREE', 1 link
    [1, 0], # Legitimate: 1 'FREE', no link
    [5, 4], # Spam: 5 'FREE', 4 links
    [8, 6], # Spam: 8 'FREE', 6 links
    [0, 2], # Legitimate: No 'FREE', 2 links
    [6, 3]  # Spam: 6 'FREE', 3 links
])
y_train = np.array([0, 0, 1, 1, 0, 1])

# 2. Instantiate the Logistic Regression Classifier object
classifier = LogisticRegression()

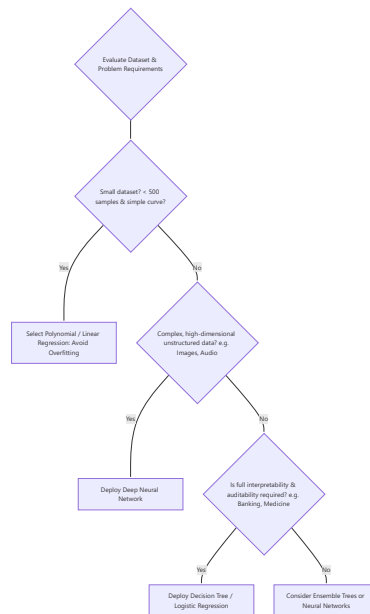
# 3. Fit model (optimizes sigmoid decision threshold parameters)
classifier.fit(x_train, y_train)

# 4. Make predictions on unseen incoming email: [4 'FREE' occurrences, 5 links]
unseen_email = np.array([[4, 5]])
predicted_class = classifier.predict(unseen_email)
predicted_probabilities = classifier.predict_proba(unseen_email)

print(f"Predicted Category: {'SPAM' if predicted_class[0] == 1 else 'HAM'}")
print(f"Confidence (Spam Probability): {predicted_probabilities[0][1] * 100:.1f}%")
```

7.4 Model Selection: Neural Networks vs Simpler Models

A common error in software automation is defaulting to deep neural networks for every problem. Software engineers must match algorithmic complexity to data volume, interpretability requirements, and computational constraints.



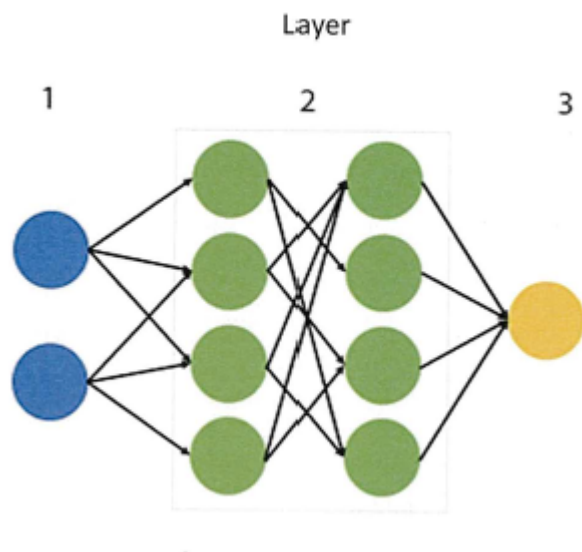
DIMENSION	SIMPLE REGRESSION (LINEAR / POLYNOMIAL)	ARTIFICIAL NEURAL NETWORKS (ANNs)
Data Requirement	Operates effectively on very small datasets (20–100 samples).	Requires large training datasets (thousands to millions of examples) to avoid overfitting.
Computational Overhead	Minimal; solves in milliseconds with low CPU and memory usage.	High; requires intensive matrix operations, long training times, and dedicated GPU accelerators.
Model Interpretability	Glass Box: Directly interpretable via explicit algebraic mathematical coefficients ($y = mx + b$).	Black Box: Difficult to explain; decisions emerge from thousands of opaque weight interactions.
Risk of Overfitting	Low when constrained to low polynomial degrees ($d \leq 3$).	Very high on small or noisy datasets unless carefully managed with dropout or regularization.

Chapter 7 Practice Questions

Multiple Choice Questions

Question 7.1 [Source: 2025 CSSA Trial, Q7]

The diagram below represents an artificial neural network:



Match each description to its correct layer number:

1. Transforms and processes data through weighted connections and activation functions: []
2. Predictions are generated based on input data and transformations: []
3. Ingests the original input data features for the network: []

Which sequence correctly identifies the layers from 1 to 3?

- (A) Layer 1; Layer 2; Layer 3
- (B) Layer 2; Layer 3; Layer 1
- (C) Layer 3; Layer 1; Layer 2
- (D) Layer 2; Layer 1; Layer 3

Question 7.2 [Source: 2025 CSSA Trial, Q10]

Which of the following machine learning models is typically used in virtual personal assistants (such as Siri or Alexa) to convert spoken sound waves into text and parse natural language intents?

- (A) Decision trees
- (B) Linear regression
- (C) Logistic regression
- (D) Neural networks

Short Answer Questions

Question 7.3 (3 marks) [Source: 2025 CSSA Trial, Q16]

Explain how an Object-Oriented Programming (OOP) approach enhances the development and automation of machine learning models.

Question 7.4 (5 marks) [Source: HSC Style Questions 2025, Q13 / Sample Trial 2025, Q11b]

A software engineer has collected a small experimental dataset of 25 observations relating chemical reaction rates (y) to catalyst concentrations (x). A scatter plot of the data shows a smooth, non-linear, upward-curving trend. Explain why a Neural Network would **not** be the best model to train on this dataset, and propose a more appropriate alternative model with justification.

Chapter 7 Marking Guidelines & Exemplars

Question 7.1 Solution

- **Correct Answer:** (B) Layer 2; Layer 3; Layer 1.
- *Rationale:* Layer 2 (Hidden) handles weighted transformations and activation functions; Layer 3 (Output) generates final predictions; Layer 1 (Input) ingests raw feature data.

Question 7.2 Solution

- **Correct Answer:** (D) Neural networks.
- *Rationale:* Virtual personal assistants rely on deep neural networks (specifically CNNs, RNNs, and Transformers) to process audio waveforms into phonemes and parse natural language semantics.

Question 7.3 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly explains how OOP structures ML models, explicitly addressing modularity, encapsulation of parameters/methods, and reusability across automated pipelines. • Provides clear technical examples (e.g. classes, <code>.fit()</code>, <code>.predict()</code>).	3
• Outlines basic OOP principles in relation to machine learning with minor omissions.	2
• States an isolated feature of OOP or machine learning.	1

Band 6 Sample Exemplar:

An Object-Oriented Programming (OOP) approach structures machine learning by encapsulating model data, hyperparameters, and algorithms inside modular, reusable classes. For example, a model class bundles internal parameters (such as connection weights and intercepts) and exposes standardized methods like `.fit()` for training and `.predict()` for inference.

This modular design allows developers to automate data-processing and training pipelines without rewriting boilerplate code. Furthermore, through inheritance and polymorphism, different algorithms can share a common interface, allowing automated CI/CD and MLOps systems to swap, test, and deploy different models interchangeably.

Question 7.4 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly explains why a neural network is unsuitable, addressing data scarcity (only 25 points), high risk of overfitting, and computational overhead. • Proposes Polynomial Regression as the alternative model with comprehensive justification linked to the smooth curvilinear data shape.	5
• Explains two limitations of neural networks on small datasets and proposes a justified alternative model.	4
• Explains one limitation of neural networks and identifies a suitable alternative model.	3
• Outlines a basic issue with neural networks or names an alternative model without detailed justification.	2
• Provides limited relevant information.	1

Band 6 Sample Exemplar:

Why a Neural Network is Unsuitable:

1. **Data Scarcity & Overfitting:** Neural networks contain dozens or hundreds of internal weights and biases that must be tuned iteratively. With only 25 data points, a neural network will easily overfit, memorizing the noise in individual samples rather than learning the true underlying chemical relationship.
2. **Excessive Complexity & Compute:** Neural networks are computationally intensive "black box" models requiring iterative backpropagation, hyperparameter tuning, and activation function configurations, which introduces unnecessary complexity for a simple single-feature problem.

Proposed Alternative Model:

- **Polynomial Regression (Degree 2 or 3).**
- **Justification:** The scatter plot exhibits a smooth, continuous, non-linear relationship between catalyst concentration and reaction rate. A low-degree polynomial regression directly models this smooth curve using an interpretable mathematical formula with few parameters, computes near-instantaneously, and avoids overfitting, providing more reliable predictions on unseen data.

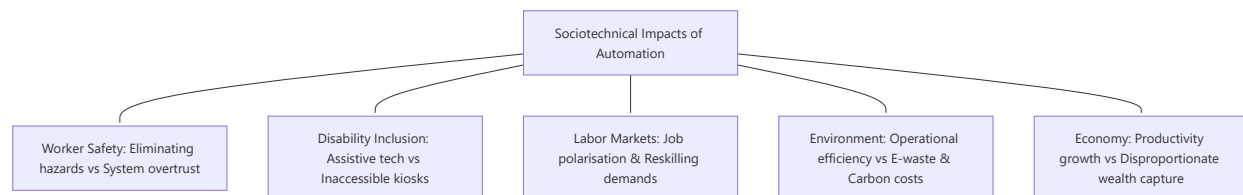
Chapter 8 - Sociotechnical Impacts, Human Behaviour & Algorithmic Bias

Syllabus Dot Points Covered:

- *Significance and impact of ML and AI* — Assess the impact of automation on the individual, society and the environment: safety of workers, people with disability, the nature and skills required for employment, production efficiency, waste and the environment, the economy and distribution of wealth
- *Significance and impact of ML and AI* — Explore by implementation how patterns in human behaviour influence ML and AI software development: psychological responses, patterns related to acute stress response, cultural protocols, belief systems
- *Significance and impact of ML and AI* — Investigate the effect of human and dataset source bias in the development of ML and AI solutions

8.1 The Broader Impacts of Software Automation

Software automation and artificial intelligence impact workplaces, economies, and natural ecosystems. Software engineers must evaluate these broader effects throughout the development lifecycle.



1. Safety of Workers

- **Positive:** Automating dangerous tasks eliminates physical workplace hazards. Robotic arms handle heavy toxic chemical manufacturing, high-temperature metal welding, and industrial pressing. Drones inspect high-voltage power lines and offshore oil rigs, protecting workers from dangerous environments.
- **Negative:** Automated workplaces introduce new risks, such as human-robot collisions in shared logistics warehouses. Additionally, automated systems can lead to **automation bias (reduced human oversight)**, where human operators place uncritical trust in automated systems, failing to notice software anomalies or sensor failures.

2. People with Disability

- **Positive:** AI-driven computer vision and speech recognition provide powerful assistive tools. Screen-reading systems with Optical Character Recognition (OCR) convert printed text into

synthetic speech for visually impaired users. Eye-tracking systems and voice-activated assistants enable individuals with severe motor impairments to control workstations and navigate software independently.

- **Negative:** Automation can also create accessibility barriers if not designed inclusively. Self-service touchscreen kiosks lacking audio jacks, tactile interfaces, or adjustable viewing angles exclude wheelchair users and individuals with visual impairments.

3. Employment and the Nature of Skills

- **Labor Market Shift:** Routine, manual, and predictable administrative roles (e.g., data-entry clerks, basic bookkeeping, repetitive assembly line work) are increasingly automated. Conversely, demand is growing for software engineers, data analysts, robotics technicians, and machine learning maintenance specialists.
- **The Digital Divide & Reskilling:** Workers displaced by automation must learn digital literacy, basic data interpretation, or software configuration skills. Older workers and individuals in regional communities often face limited access to training resources, exacerbating regional unemployment.

4. Production Efficiency, Waste, and the Environment

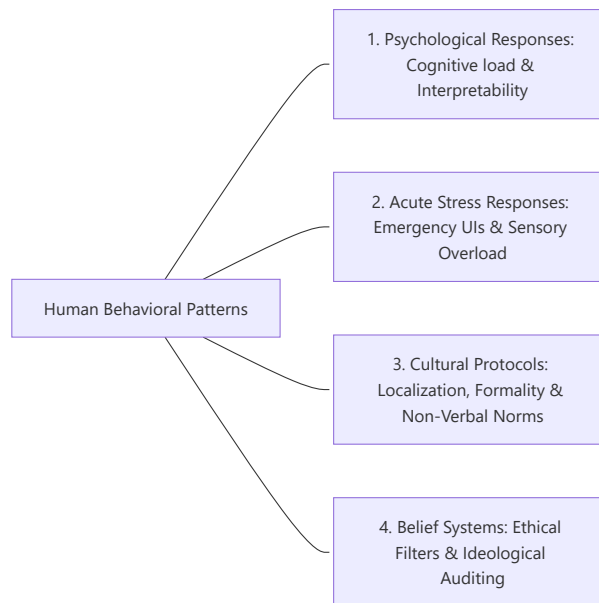
- **Efficiency & Waste Reduction:** Automated systems optimize material cutting, chemical balancing, and just-in-time manufacturing, reducing industrial scrap and waste. In agriculture, automated sensor networks and camera-equipped drones target pesticide spraying directly at weeds, reducing chemical runoff into local waterways.
- **Environmental Costs:** Training large deep-learning models and running global AI cloud data centers 24/7 consumes massive amounts of electricity and fresh water for cooling. Furthermore, rapid hardware upgrade cycles drive high volumes of **electronic waste (e-waste)**, which contains hazardous heavy metals that pollute landfills if improperly disposed of.

5. Economy and the Distribution of Wealth

- **Economic Growth vs Wealth Concentration:** Automation boosts manufacturing productivity and corporate profit margins by lowering labor costs. However, these financial gains often concentrate disproportionately among technology platform owners and intellectual property holders. Meanwhile, wages in manual service jobs may stagnate, contributing to economic inequality.

8.2 Human Behavioural Influences on AI/ML Development

Automated systems interact with human users whose decisions are shaped by emotion, stress, cultural backgrounds, and values. Effective software engineering accounts for these behavioural patterns in both user interface design and algorithmic logic:



1. Psychological Responses

Users experience cognitive fatigue and frustration when interacting with opaque, confusing automated systems.

- **Design Considerations:** Engineers use **progressive disclosure**—presenting essential information first while keeping advanced settings easily accessible—to manage cognitive load.
- **Transparency:** In high-stakes environments (e.g., automated medical triage or loan rejections), software engineers integrate explainability tools (such as feature importance charts) so users can understand *why* the system made a particular recommendation, helping build trust.

2. Acute Stress Response

Under extreme pressure, fear, or sensory overload (e.g., emergency dispatches, industrial control room alerts, medical emergencies), human decision-making narrows, reaction times drop, and users are prone to operational errors.

- **Design Considerations:** Automated user interfaces for high-stress systems must minimize cognitive friction. They feature high-contrast color coding (e.g., red for immediate emergency stop), large touch targets, simplified one-click confirmation dialogs, and automated voice prompts rather than dense blocks of text.

3. Cultural Protocols

Societal norms, idioms, formality levels, and non-verbal gestures vary widely across cultures.

- **Design Considerations:** Machine learning conversational agents must be localized with culturally sensitive language models. An assistant configured for formal business communication in Japan must implement appropriate honorifics, whereas an assistant in Australia might adopt a more casual tone. Training datasets must include diverse global demographics to prevent virtual assistants from misinterpreting regional accents or slang.

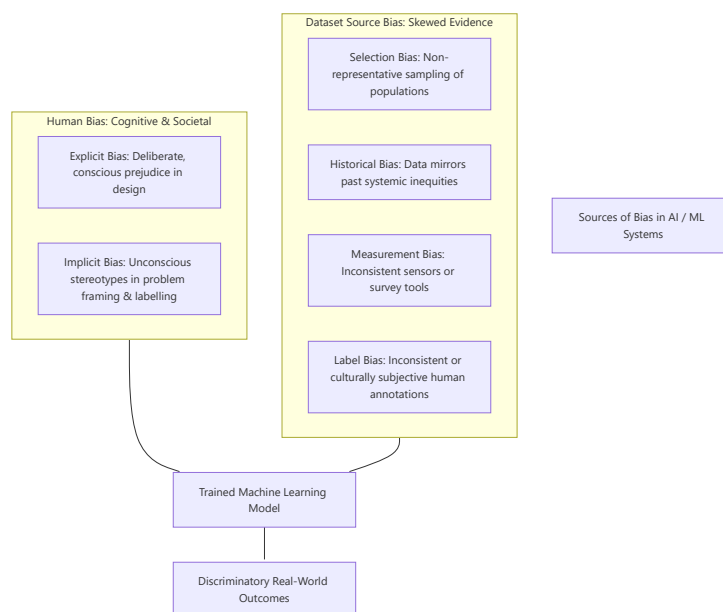
4. Belief Systems

Users hold diverse ethical, moral, and religious values that influence their expectations of software behavior.

- **Design Considerations:** Automated recommendation engines and content moderation systems must include ethical filters and user controls, allowing individuals to opt out of content that conflicts with their religious or moral values. Furthermore, algorithms in sensitive domains (such as criminal justice risk assessments or hiring tools) must be audited to ensure they do not enforce biased ideological assumptions.

8.3 Algorithmic, Human, and Dataset Source Bias

When machine learning models generate unfair or discriminatory predictions, the root cause is rarely the mathematical algorithm itself. Rather, models reflect and amplify the biases present in their training data and development teams.



Types of Human Bias

- **Explicit (Conscious) Bias:** Occurs when developers intentionally build subjective preferences or exclusionary criteria directly into the system's requirements or data collection.
- **Implicit (Unconscious) Bias:** Occurs when developers' unexamined social stereotypes influence how they frame problems, select features, or annotate training data. For example, developers training an automated hiring tool might unconsciously label resumes containing sports metaphors as "high leadership," inadvertently favoring male applicants.

Types of Dataset Source Bias

- **Selection Bias:** Occurs when the training dataset is drawn from a narrow or skewed population that fails to reflect real-world demographic diversity. For example, if a facial-recognition security system

is trained on an image database consisting of 85% light-skinned male faces, the model will exhibit high error rates when attempting to identify women and individuals with darker skin tones.

- **Historical Bias:** Occurs when the data accurately reflects the past, but the past contains systemic social inequities. For example, training a predictive policing algorithm on historical arrest records will lead the model to direct police patrols to historically over-policed minority neighborhoods, creating a self-fulfilling feedback loop.
- **Label Bias:** Occurs when human annotators apply subjective or inconsistent labels to training examples (e.g., annotating pictures of diverse cultural clothing as "unprofessional").

Strategies to Detect and Mitigate Algorithmic Bias

1. **Diverse and Representative Sampling:** Actively assembling training datasets that reflect the demographic diversity of the real-world operational environment.
 2. **Algorithmic Fairness Audits:** Testing model accuracy across separate demographic cohorts (e.g., verifying that false positive rates for loan defaults are equal across all racial and gender sub-groups).
 3. **Data Pre-processing & Sanitisation:** Stripping direct and proxy demographic identifiers (e.g., removing postal codes that correlate closely with racial demographics) and re-weighting underrepresented samples.
 4. **Multidisciplinary Engineering Teams:** Involving diverse engineers, ethicists, and domain specialists throughout the development lifecycle to identify blind spots early.
-

Chapter 8 Practice Questions

Multiple Choice Questions

Question 8.1 [Source: Claritas Original]

A hospital implements an automated clinical diagnosis tool for skin cancer. When tested across different patient groups, the system shows an 18% higher misdiagnosis rate for patients with darker skin tones. Which source of bias is the primary cause of this disparity?

- (A) Measurement bias caused by inconsistent hospital network bandwidth
- (B) Selection bias due to training datasets composed predominantly of lighter skin images
- (C) Algorithmic underfitting caused by low polynomial regression degrees
- (D) Verification errors caused by using POP3 rather than IMAP

Question 8.2 [Source: 2025 HSC Exam, Q24]

A construction company introduces an AI safety monitoring system using wearable sensors and overhead cameras to track worker movements in real time, detecting unsafe proximity to heavy machinery. What is a significant negative psychosocial impact this system could have on workers?

- (A) Increased likelihood of mechanical machine breakdowns
- (B) Invasive surveillance causing acute workplace stress and anxiety
- (C) Inability of the AI model to communicate over HTTPS
- (D) Complete elimination of skilled robotics maintenance roles

Short Answer Questions

Question 8.3 (5 marks) [Source: 2025 HSC Exam, Q24]

A construction company introduces a safety monitoring system that uses artificial intelligence. Cameras and wearable sensors track worker movements and environmental conditions, detecting unsafe behaviors such as standing too close to operating machinery, entering restricted zones, and failing to wear personal protective equipment (PPE). Real-time alerts are sent to site supervisors, and long-term logs are stored for compliance analysis.

Discuss the potential impacts of this AI monitoring system on workers.

Question 8.4 (6 marks) [Source: 2025 HSC Exam, Q27 / HSC Style Questions 2025, Q22]

Explain how human and dataset source bias can affect the accuracy and fairness of automated medical imaging systems. In your response, outline TWO technical strategies software engineers can implement to minimize these biases.

Chapter 8 Marking Guidelines & Exemplars

Question 8.1 Solution

- **Correct Answer:** (B) Selection bias due to training datasets composed predominantly of lighter skin images.
- **Rationale:** The diagnostic failure stems from selection bias in the training data—an unrepresentative dataset that lacks sufficient diversity across all skin pigmentation categories.

Question 8.2 Solution

- **Correct Answer:** (B) Invasive surveillance causing acute workplace stress and anxiety.
- **Rationale:** Continuous, automated biometric tracking can create feelings of invasive workplace surveillance and acute stress, particularly if workers feel monitored during breaks or fear punitive actions for false-positive alerts.

Question 8.3 Marking Guide

MARKING CRITERIA	MARKS
• Comprehensively discusses both positive AND negative impacts of the AI monitoring system on workers, referencing the scenario directly. • Covers physical safety, reduced human error, privacy concerns, stress from surveillance, and the risk of false-positive alarms.	5
• Discusses both positive and negative impacts on workers with sound detail.	4
• Outlines some impacts on workers, focusing mainly on one side (either positive or negative).	3

MARKING CRITERIA	MARKS
• Mentions an isolated positive or negative impact on workers.	2
• Provides limited relevant information.	1

Band 6 Sample Exemplar:

Positive Impacts:

The AI safety monitoring system significantly improves physical workplace safety by reducing human error. Wearable sensors and computer-vision cameras provide real-time proximity alerts when a worker inadvertently steps into a blind spot near heavy excavators, allowing supervisors to intervene immediately and prevent catastrophic industrial accidents. It also automates PPE compliance monitoring, ensuring all workers wear helmets and high-visibility gear in hazardous zones, fostering a proactive workplace safety culture.

Negative Impacts:

However, constant tracking can have adverse psychosocial impacts on workers. Continuous monitoring creates feelings of invasive workplace surveillance, causing acute stress and anxiety—especially if workers feel their physical movements are scrutinized during breaks or private moments. Furthermore, system false alarms (e.g. misidentifying safe proximity as a hazard) can create alert fatigue, causing frustration or distracting workers at critical operational moments. Workers may also worry that aggregated movement logs could be used unfairly during performance reviews.

Question 8.4 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly explains how human bias (explicit/implicit) AND dataset source bias (selection/label) compromise medical imaging accuracy and lead to discriminatory health outcomes. • Proposes TWO distinct, technically sound mitigation strategies (e.g. demographic cohort testing, synthetic data augmentation, blind expert annotation, bias audits).	6
• Explains the effects of bias on medical imaging and outlines two mitigation strategies with minor omissions.	5
• Outlines sources of bias and describes at least one technical mitigation strategy.	3–4
• Identifies basic concepts of bias in machine learning.	2
• Provides limited relevant information.	1

Band 6 Sample Exemplar:**Impact of Bias in Medical Imaging:**

Automated medical imaging systems (such as convolutional neural networks diagnosing melanoma or chest X-rays) learn visual features directly from training datasets. If these datasets suffer from **selection bias**—for instance, if dermatological image libraries primarily feature light-skinned patients—the model fails to learn how pathologies present across darker skin pigmentations.

Additionally, **human label bias** occurs when medical annotators inconsistently categorize ambiguous edge cases based on subjective judgment. These biases lead to reduced diagnostic accuracy and higher false-negative rates for underrepresented demographic cohorts, risking delayed treatment and reinforcing healthcare inequities.

Technical Mitigation Strategies:

1. **Stratified, Representative Data Sourcing & Augmentation:** Software engineers must establish strict data curation pipelines that assemble balanced, demographically diverse training datasets across varied ages, ethnicities, and imaging hardware. If real-world clinical samples are scarce for specific cohorts, engineers can use generative techniques or data augmentation to balance sample distributions before model training.
 2. **Demographic Disaggregated Model Audits:** During validation, the model must be evaluated using performance metrics (such as sensitivity and false-negative rates) broken down across individual demographic groups. If the model exhibits an accuracy disparity exceeding acceptable clinical thresholds on a specific subgroup, the system is prevented from production deployment until the model is retrained and debiased.
-

Chapter 9 - Secure Design Principles, the SDLC & Cryptographic Defences

Syllabus Dot Points Covered:

- *Designing software* — Describe the benefits of developing secure software: data protection, minimising cyber attacks and vulnerabilities
- *Designing software* — Interpret and apply fundamental software development steps to develop secure code: requirements definition, determining specifications, design, development, integration, testing and debugging, installation, maintenance
- *Designing software* — Describe how the capabilities and experience of end users influence the secure design features of software
- *Developing secure code* — Explore fundamental software design security concepts when developing programming code: confidentiality, integrity, availability, authentication, authorisation, accountability
- *Developing secure code* — Apply security features incorporated into software including data protection, security, privacy and regulatory compliance
- *Developing secure code* — Use and explain the contribution of cryptography and sandboxing to the 'security by design' approach in the development of software solutions
- *Developing secure code* — Use and explain the 'privacy by design' approach in the development of software solutions: proactive not reactive approach, embed privacy into design, respect for user privacy

9.1 The Strategic Necessity of Secure Software Architecture

In traditional software development, security was frequently treated as an afterthought—a perimeter firewall or an ad-hoc patch applied after an application was already constructed. **Secure software architecture** fundamentally shifts security to the foundation of the engineering lifecycle, embedding defensive controls into every structural component.



1. Data Protection and the CIA Triad

The core objective of software security is safeguarding data assets across all states: **at rest** (stored on disk or database), **in transit** (traversing network interfaces), and **in use** (residing in memory during execution).

- **Confidentiality:** Preventing unauthorized disclosure of sensitive information.
- **Integrity:** Protecting information against unauthorized modification, deletion, or tampering.
- **Availability:** Ensuring legitimate users maintain timely, uninterrupted access to critical software services.

2. Minimising Cyber Attacks and Entry Points

Every exposed input box, open port, and API endpoint constitutes an **attack surface**. Secure architecture adheres to the principle of least privilege, eliminates single points of failure, and implements automated defense layers, drastically reducing the window of opportunity for exploits like SQL injection, buffer overflows, and remote code execution (RCE).

3. Business Continuity and Legal Compliance

System outages cause immense financial damage. In Australia, software architectures processing personal identifiable information (PII) must comply with federal statutory frameworks, notably the **Privacy Act 1988 (Cth)** and the **Notifiable Data Breaches (NDB) scheme**. Secure systems ensure that services remain resilient during active distributed denial-of-service (DDoS) assaults and protect enterprises from catastrophic regulatory fines.

9.2 The Secure Software Development Life Cycle (SSDLC)

Securing software requires applying tailored security practices across all eight fundamental stages of software development:

DEVELOPMENT STAGE	CORE SECURITY ACTIVITIES & ENGINEERING CONTROLS
1. Requirements Definition	• Identify compliance mandates (e.g. Privacy Act, GDPR). • Conduct initial risk assessments and threat identification. • Define explicit security objectives (e.g. multi-factor authentication).
2. Determining Specifications	• Specify Role-Based Access Control (RBAC) matrix levels. • Define exact input schemas, character lengths, and regex boundary rules. • Establish throughput rate-limits for denial-of-service defense.
3. Design	• Apply the Principle of Least Privilege and Fail-Safe Defaults. • Select approved cryptographic algorithms (e.g. AES-256, SHA-256). • Architect network sandboxing, reverse proxies, and DMZ enclaves.
4. Development	• Adhere to secure coding standards (e.g. OWASP guidelines, CERT rules). • Implement parameterised SQL queries and strict input sanitisation. • Implement secure error handling (suppress raw stack traces).
5. Integration	• Audit third-party packages and dependencies for known CVE vulnerabilities. • Enforce mutual TLS (mTLS) token-based inter-service communication. • Strip debug backdoors, hardcoded API keys, and staging endpoints.
6. Testing and Debugging	• Run Static Application Security Testing (SAST) on source code repositories. • Execute Dynamic Application Security Testing (DAST) on running instances. • Conduct ethical penetration testing and protocol fuzzing.

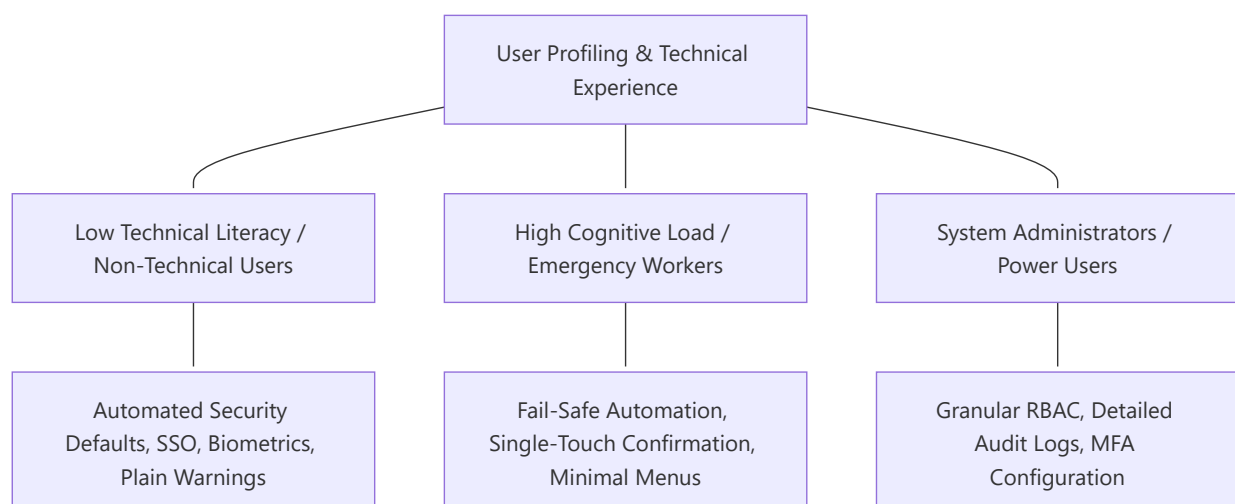
DEVELOPMENT STAGE	CORE SECURITY ACTIVITIES & ENGINEERING CONTROLS
7. Installation	• Apply secure baseline server configurations (disable default root accounts). • Configure host firewalls, close unused ports, and enforce HTTPS. • Verify cryptographically signed installation packages and binaries.
8. Maintenance	• Continuously monitor system telemetry and centralized immutable audit logs. • Deploy signed security hotfixes and patch dependency vulnerabilities. • Execute periodic disaster recovery and automated failover drills.

Claritas Exam Tip: Requirements Definition vs Design

In HSC examination questions asking where security first impacts a project, the correct stage is **Requirements Definition** (identifying potential threats, regulatory constraints, and security objectives). However, if the question asks where specific cryptographic algorithms or architectural patterns are *selected*, the correct stage is **Design**.

9.3 End-User Capabilities and Secure UX Design

Security mechanisms that fail to account for human psychology and technical competency will inevitably be bypassed by end users seeking operational convenience. Engineers must tailor security features to user capabilities:



USER PROFILE / TRAIT	OPERATIONAL VULNERABILITY	SECURE SOFTWARE DESIGN COUNTERMEASURE
Low Technical Literacy	Prone to selecting weak passwords ("password123") and susceptible to social engineering.	Enforce automated password managers, single sign-on (SSO), or passkeys; display plain-language security warnings instead of complex technical alerts.
Cognitive Fatigue / High Stress	Quick to accept permission prompts without reading; prone to misdirected data entry.	Implement progressive disclosure; require explicit confirmation dialogues for high-risk actions (e.g. deleting databases); use clear color coding.
Shared Hardware Access	Forgets to log out when accessing systems from public library or school computers.	Enforce aggressive session timeouts (e.g. 5 minutes of inactivity) and auto-terminate sessions when browser tabs close.
Motor or Visual Impairments	Inability to complete visual image CAPTCHAs or manipulate micro-checkboxes.	Provide W3C WAI-compliant multi-modal authentication (e.g. accessible audio CAPTCHAs, biometric tokens, full keyboard navigation).

9.4 Core Information Security Principles: The CIA + AAA Framework

Software design security revolves around six foundational pillars:

1. **Confidentiality:** Restricts data access strictly to authorized entities.
 - *Code Exemplar:* Encrypting database columns storing credit card numbers using AES-256, and setting the `HttpOnly` and `Secure` flags on session cookies to prevent JavaScript exposure.
2. **Integrity:** Ensures data remains mathematically identical to its authorized state.
 - *Code Exemplar:* Generating SHA-256 hashes of operating system executable patches to verify that files have not suffered tampering during distribution.
3. **Availability:** Guarantees systems remain operational despite hardware failures or malicious assaults.
 - *Code Exemplar:* Implementing API rate limiting (e.g. maximum 60 requests per IP per minute) to prevent Denial of Service (DoS) server exhaustion.
4. **Authentication (AuthN):** Confirms that an entity is genuinely who they claim to be.

- *Code Exemplar:* Verifying salted password hashes via `bcrypt.checkpw()`, supplemented by time-based one-time passwords (TOTP).
5. **Authorisation (AuthZ):** Determines what actions an authenticated entity is permitted to perform.
- *Code Exemplar:* Intercepting an API request to check whether `user.role == 'Admin'` before granting execution privileges for `DROP TABLE`.
6. **Accountability:** Links every system transaction unambiguously to an authorized identity.
- *Code Exemplar:* Writing immutable, cryptographically signed log entries recording user ID, IP address, timestamp, and action description to a write-only log server.

9.5 'Security by Design' vs 'Privacy by Design'

While often used interchangeably in non-technical discussions, software engineers draw a clear distinction between these two philosophies:



Syllabus Definition

- **Security by Design:** An engineering approach where security controls, threat models, and vulnerability mitigations (such as cryptography and sandboxing) are built into the software architecture from its inception, rather than bolted on after deployment.
- **Privacy by Design (PbD):** A development philosophy that embeds regulatory privacy protections, data minimization, and user control directly into software defaults. It is governed by three mandatory tenets:
 1. **Proactive not Reactive:** Anticipating and preventing privacy-invasive events before they occur, rather than offering apologies after a breach.
 2. **Embed Privacy into Design:** Making privacy an essential, default capability of the core architecture without compromising functional capabilities.
 3. **Respect for User Privacy:** Keeping software user-centric by providing transparent consent notices, defaults set to maximum privacy, and tools to inspect or delete stored data.

The Role of Cryptography in Security by Design

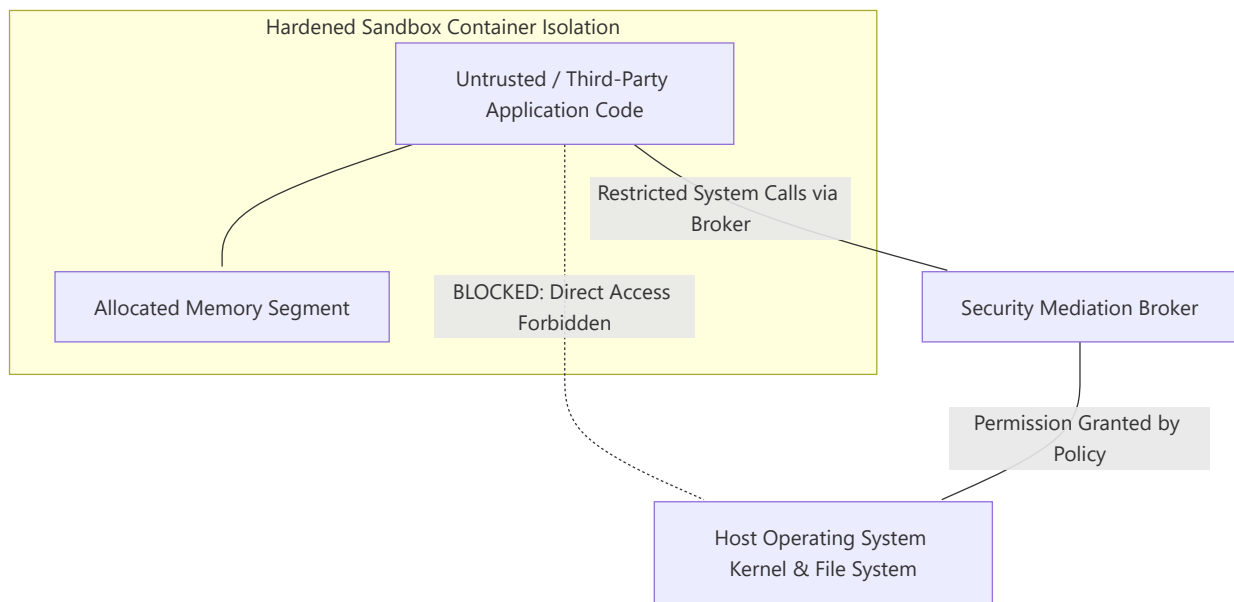
Cryptography provides mathematical guarantees of confidentiality, authenticity, and non-repudiation:

- **Symmetric Encryption (e.g. AES-GCM):** Encrypts bulk application databases and files at rest using a single fast key.

- **Asymmetric Encryption (e.g. RSA, ECC):** Facilitates secure key exchanges across untrusted internet networks.
- **Cryptographic Hashing (e.g. SHA-256):** Provides one-way verification digests for passwords and system files.

The Role of Sandboxing in Security by Design

Sandboxing isolates executing software components within tightly constrained logical boundaries. The sandbox limits the program's access to CPU cycles, system memory, file systems, and network sockets:



- **Desktop & Mobile Operating Systems:** Mobile operating systems (such as iOS and Android) assign each application its own isolated user ID and file directory sandbox. A downloaded game cannot read stored messages from a banking app or access the microphone without explicit operating system broker prompts.
- **Web Browsers:** JavaScript executed inside a browser runs within a strict client-side sandbox. The script is strictly forbidden from reading the local user's hard drive files or modifying system operating files.

Chapter 9 Practice Questions

Multiple Choice Questions

Question 9.1 [Source: 2025 CSSA Trial, Q3]

What is the first step in the software development life cycle that impacts security?

- (A) Design
- (B) Maintenance
- (C) Requirements definition
- (D) Testing

Question 9.2 [Source: 2025 CSSA Trial, Q4]

An employee at an accounting firm accidentally sent a customer's personal financial details to the wrong client via email. Although the data was not altered, it was exposed to someone who should not have access to it. Which security principle was primarily compromised?

- (A) Authorisation
- (B) Availability
- (C) Confidentiality
- (D) Integrity

Short Answer Questions**Question 9.3 (3 marks)** [Source: 2025 CSSA Trial, Q14]

Explain the difference between the 'security by design' approach and the 'privacy by design' approach in software engineering.

Question 9.4 (3 marks) [Source: HSC Familiarisation Questions, Q6]

A new operating system is to be developed for a mobile phone. The operating system needs to allow users to install a variety of third-party applications, such as social media platforms and games. Discuss the use of sandboxing in this mobile operating system.

Chapter 9 Marking Guidelines & Exemplars**Question 9.1 Solution**

- **Correct Answer:** (C) Requirements definition.
- *Rationale:* Security begins at requirements definition, where threat modeling, regulatory mandates, and security objectives are first established.

Question 9.2 Solution

- **Correct Answer:** (C) Confidentiality.
- *Rationale:* Confidentiality ensures that data is accessible only to authorized entities. Disclosing private records to an unauthorized recipient breaches confidentiality.

Question 9.3 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly explains the difference between Security by Design and Privacy by Design. • Clearly articulates that Security by Design focuses on protecting systems and data against technical vulnerabilities and cyberattacks, whereas Privacy by Design focuses on embedding user rights, data minimization, and regulatory compliance into architectural defaults.	3

MARKING CRITERIA	MARKS
• Demonstrates a sound understanding of both approaches with minor gaps in technical contrast.	2
• Demonstrates basic understanding of either approach in isolation.	1

Band 6 Sample Exemplar:

'Security by Design' is an architectural philosophy that integrates defensive measures throughout the engineering process to protect systems and data from unauthorized access, exploitation, and cyber attacks. It focuses on technical resilience using tools like cryptography, sandboxing, and input validation.

In contrast, 'Privacy by Design' focuses on embedding user rights and data governance into the software from its inception. It emphasizes collecting the minimum necessary personal data, ensuring private default configurations, and providing users with transparent control over how their personal information is processed, satisfying statutory privacy regulations. While Security by Design protects systems from external attacks, Privacy by Design respects user privacy rights by design.

Question 9.4 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly discusses the use of sandboxing in the mobile operating system, addressing both advantages (process isolation, preventing unauthorized resource access, fault tolerance) and practical disadvantages (resource overhead, constrained cross-app interactions).	3
• Outlines advantages or features of sandboxing within an operating system environment.	2
• Identifies a basic characteristic of sandboxing.	1

Band 6 Sample Exemplar:

Sandboxing provides essential security by executing each third-party application within an isolated container, restricting access to system memory, camera hardware, and other apps' local data. If a downloaded mobile game contains malicious code or suffers a memory leak, the sandbox prevents it from reading banking app credentials or crashing the mobile operating system.

However, sandboxing introduces technical trade-offs: running isolated environments consumes additional system memory, and engineers must build secure inter-process communication (IPC) permission brokers to allow safe sharing of photos or files between applications.

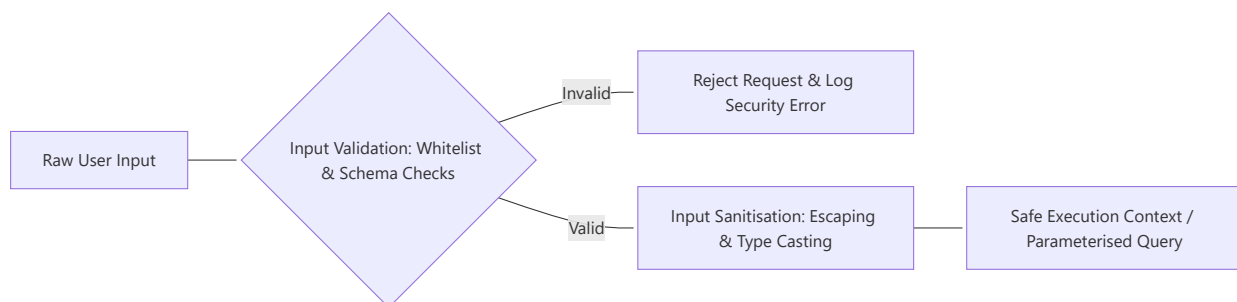
Chapter 10 - Vulnerability Mitigation, User Controls & Secure Code

Syllabus Dot Points Covered:

- *Developing secure code* — Design, develop and implement code using defensive data input handling practices, including input validation, sanitisation and error handling
- *Developing secure code* — Design, develop and implement a safe application programming interface (API) to minimise software vulnerabilities
- *Developing secure code* — Design, develop and implement code considering efficient execution for the user: memory management, session management, exception management
- *Developing secure code* — Design, develop and implement secure code to minimise vulnerabilities in user action controls: broken authentication and session management, cross-site scripting (XSS) and cross-site request forgery (CSRF), invalid forwarding and redirecting, race conditions
- *Developing secure code* — Design, develop and implement secure code to protect user file and hardware vulnerabilities from file attacks and side channel attacks

10.1 Defensive Data Input Handling

In secure software engineering, the fundamental rule of input handling is: **all external input is hostile until proven otherwise**. Attackers manipulate input vectors to break execution flow, manipulate databases, or run arbitrary commands.



1. Input Validation vs Input Sanitisation

- **Input Validation:** Inspects whether an incoming input conforms strictly to predetermined structural criteria (data type, character length, acceptable numeric range, regular expression patterns). If the input violates the schema, it is rejected entirely.
 - *Example:* Verifying that an Australian mobile number contains only numeric digits, begins with `04`, and is exactly 10 digits long.

- **Input Sanitisation:** Cleanses, strips, or encodes dangerous characters from accepted input so that the string can be processed safely without altering its underlying meaning.
 - *Example:* Converting special HTML characters such as `<` and `>` into their safe entity equivalents (`<` and `>`), preventing user-supplied strings from executing as scripts in a browser.

2. Secure Error and Exception Handling

Default web frameworks often output verbose stack traces when an uncaught exception occurs. These stack traces leak critical infrastructure details to attackers, including database versions, operating system directories, and library dependencies.

- **Defensive Strategy:** Implement global `try-catch` blocks that catch low-level errors, write detailed debugging traces to an internal, encrypted server log, and return clean, generic error messages to the client (e.g. `"An unexpected error occurred. Please reference Ticket #4029."`).

```
# Defensive Data Input Handling in Python
import re

def process_user_age_input(raw_input_string):
    """Demonstrates input validation, type casting, and secure error handling."""
    # 1. Input Validation: Whitelisting regex ensuring input consists ONLY of digits
    if not re.fullmatch(r"^[0-9]{1,3}$", raw_input_string):
        # Graceful failure: return a safe flag without exposing internal logic
        return False, "Validation Error: Age must be a positive integer between 0 and 150."

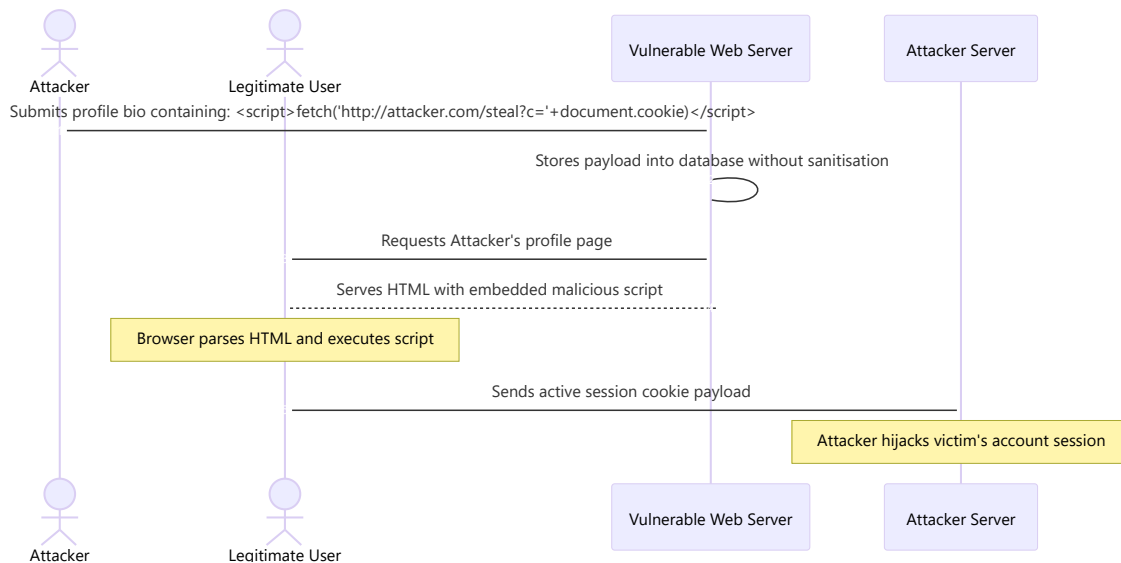
    try:
        age = int(raw_input_string)
        # Boundary validation
        if 0 <= age <= 150:
            return True, age
        else:
            return False, "Validation Error: Age out of valid physical range."
    except ValueError:
        # Secure exception handling preventing application crashes
        return False, "Processing Error: Malformed input data."
```

10.2 Mitigating Vulnerabilities in User Action Controls

The Open Worldwide Application Security Project (OWASP) catalogs the most dangerous recurring application vulnerabilities. Engineers must understand how to identify and remediate these flaws in code.

1. Cross-Site Scripting (XSS)

XSS occurs when an application injects unvalidated, unsanitised user input directly into an HTML page sent to other users, allowing malicious JavaScript to execute within victims' browsers.



• XSS Remediation:

1. **Context-Aware Output Encoding:** Escape all user-supplied variables before rendering them in the DOM (`&` → `&` , `<` → `<` , `"` → `"`).
2. **Content Security Policy (CSP):** Configure HTTP response headers to restrict which domains can execute scripts:

```
Content-Security-Policy: default-src 'self'; script-src 'self'
https://trustedscripts.com;
```

2. Cross-Site Request Forgery (CSRF)

CSRF occurs when an attacker tricks an authenticated user's browser into transmitting unauthorized commands to a vulnerable website where the user is currently logged in. Because browsers automatically include ambient authentication credentials (such as cookies), the server cannot distinguish between legitimate user actions and forged requests.

• CSRF Remediation:

- **Anti-CSRF Tokens:** The server issues a cryptographically random, unpredictable token stored in the user's session. Every state-changing form submission must submit this matching token in its payload.
- **SameSite Cookie Flags:** Setting `SameSite=Strict` on session cookies ensures the browser will never attach authentication cookies to cross-site requests.

3. Broken Authentication & Session Management

Occurs when session tokens are predictable, lack expiration, or are exposed over unencrypted connections.

- **Mitigations:** Generate cryptographically secure pseudorandom numbers (CSPRNG) for session tokens (e.g. 128-bit UUIDs); enforce inactivity timeouts; invalidate tokens immediately upon logout; and prevent credential enumeration by returning generic error messages (e.g. `"Invalid username or password"`).

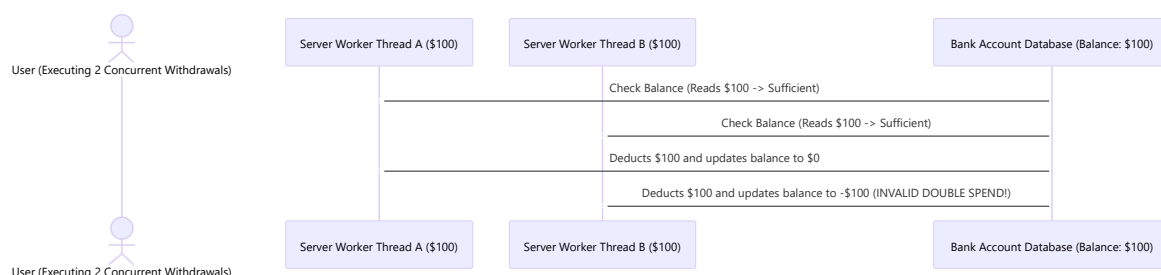
4. Invalid Forwarding and Open Redirects

Applications frequently accept URL parameters to redirect users after login (e.g. `login.php?redirect=dashboard.php`). If untrusted input is passed directly into redirection functions, attackers can craft links sending victims to credential-harvesting phishing clones.

- **Mitigation:** Enforce an internal **whitelisting check**, only permitting relative paths or registered domain names.

5. Race Conditions & Time-of-Check to Time-of-Use (TOCTOU)

A race condition occurs when concurrent threads or processes read and write shared system memory or database state simultaneously, leading to inconsistent or insecure results.



- **Mitigations:**
 - **Database Row Locking:** Wrap transactions in atomic blocks using `SELECT FOR UPDATE` to lock records until the transaction completes.
 - **Mutexes & Semaphores:** Use synchronization primitives in multithreaded code to ensure critical sections execute serially.

10.3 Defending Files, Hardware & APIs

Protecting Against File Attacks

When web applications accept file uploads (e.g. user profile photos), attackers may attempt to upload executable scripts (e.g. `shell.php` or `malware.exe`).

1. **Re-encode and Strip Metadata:** Re-render incoming images using image-processing libraries to strip dangerous executable payloads and EXIF metadata.
2. **Restrict Storage Permissions:** Store uploaded files outside the public web root in dedicated cloud storage buckets (e.g. AWS S3) with operating system execution permissions removed (`chmod 644`).
3. **MIME-Type Whitelisting:** Inspect the raw magic bytes of the file rather than trusting the user-supplied file extension.

Side-Channel Attacks

A side-channel attack extracts secret cryptographic keys or sensitive data by measuring physical hardware side-effects—such as execution time, power consumption patterns, or acoustic emissions—rather than attacking mathematical flaws in code.

```
# VULNERABLE: Variable-time string comparison (Timing Side-Channel Attack)
def insecure_verify_pin(user_pin, actual_pin):
    # Fails immediately on first mismatch -> leaks character position via execution time!
    if len(user_pin) != len(actual_pin):
        return False
    for i in range(len(user_pin)):
        if user_pin[i] != actual_pin[i]:
            return False
    return True

# SECURE: Constant-time string comparison preventing timing side-channel attacks
import hmac

def secure_verify_pin(user_pin, actual_pin):
    # Compares all bytes regardless of where differences occur, eliminating timing leaks
    return hmac.compare_digest(user_pin, actual_pin)
```

Safe Application Programming Interface (API) Engineering

1. **Token-Based Stateless Authentication:** Secure all API endpoints using cryptographically signed JSON Web Tokens (JWT) or OAuth 2.0 bearer tokens transmitted over HTTPS.
2. **Strict Payload Validation:** Validate incoming JSON bodies against a defined schema, rejecting unexpected fields to prevent **Mass Assignment vulnerabilities**.
3. **Rate Limiting & Throttling:** Implement sliding-window rate limiters to prevent brute-force credential stuffing and denial of service.

10.4 Engineering Efficient & Secure Execution

Security and performance must be engineered in tandem. Poorly optimized code can introduce severe security vulnerabilities:

- **Memory Management:** In languages like C, neglecting bounds checking causes buffer overflows, allowing attackers to overwrite execution instruction pointers. In managed languages like Python, unclosed database handles and orphaned global collections cause memory leaks, leading to Out-Of-Memory (OOM) crashes that destroy application **availability**.
 - **Session Lifecycle Optimization:** Storing complete user profiles in web server memory exhausts server RAM under high concurrency. Efficient architecture stores minimal session tokens client-side in encrypted cookies, maintaining session data in fast, external memory stores (e.g. Redis) configured with automated Time-To-Live (TTL) expiration.
-

Chapter 10 Practice Questions

Multiple Choice Questions

Question 10.1 [Source: 2025 CSSA Trial, Q5]

An attacker creates the following URL and sends it to a victim:

```
http://example.com/search?q=<script>alert('You have been hacked!');</script>
```

When the victim clicks on the link, the server reflects the malicious script back into the search results page, and it executes in the victim's browser. What type of vulnerability does this represent?

- (A) Cross-Site Request Forgery (CSRF)
- (B) Cross-Site Scripting (XSS)
- (C) Invalid forwarding and redirecting
- (D) SQL Injection

Question 10.2 [Source: 2025 HSC Exam, Q8]

File locking prevents multiple users from editing a file at the same time. When a shared file is opened by a user, file locking ensures that only this user can make changes to the file while it is locked. Which of the following does file locking minimize?

- (A) Race conditions
- (B) Invalid redirection
- (C) Cross-site scripting
- (D) Authentication issues

Short Answer Questions

Question 10.3 (2 marks) [Source: Sample Trial 2025, Q7]

Describe the difference between input validation and input sanitisation as defensive data input handling practices, providing an example of each.

Question 10.4 (4 marks) [Source: HSC Style Questions 2025, Q15b]

Discuss TWO software techniques that can prevent Cross-Site Scripting (XSS), and TWO techniques that can prevent Race Conditions.

Chapter 10 Marking Guidelines & Exemplars

Question 10.1 Solution

- **Correct Answer:** (B) Cross-Site Scripting (XSS).
- *Rationale:* The application accepts unvalidated script input in a URL query parameter and reflects it back into the browser DOM, executing unauthorized JavaScript. This is a classic Reflected Cross-Site Scripting exploit.

Question 10.2 Solution

- **Correct Answer:** (A) Race conditions.
- *Rationale:* Race conditions occur when concurrent processes access and modify shared state simultaneously. File locking ensures operations occur sequentially, preventing conflicting concurrent updates.

Question 10.3 Marking Guide

MARKING CRITERIA	MARKS
• Accurately describes the distinction between input validation and input sanitisation. • Provides a valid, illustrative technical example for each practice.	2
• Describes one practice accurately or provides examples without clear distinction.	1

Band 6 Sample Exemplar:

Input validation evaluates incoming data against a strict schema (checking type, length, or range) and rejects the input entirely if it fails to conform (e.g. verifying an Australian post code contains exactly four numeric digits).

Input sanitisation accepts the input but cleanses or escapes potentially dangerous characters before processing, preventing code execution (e.g. encoding `<script>` tags to `<script>` to neutralize Cross-Site Scripting attacks).

Question 10.4 Marking Guide

MARKING CRITERIA	MARKS
- Comprehensively discusses TWO techniques to prevent XSS (e.g. context-aware output encoding, Content Security Policy headers). - Comprehensively discusses TWO techniques to prevent Race Conditions (e.g. database row locking via transactions, using thread mutexes/atomic variables).	4
Discusses three techniques across the two categories, or thoroughly covers one category with an outline of the second.	3
Outlines two techniques across either category.	2
States an isolated security technique.	1

Band 6 Sample Exemplar:

Techniques to Prevent Cross-Site Scripting (XSS):

- Context-Aware Output Encoding:** All user-controlled variables rendered in the HTML Document Object Model must be encoded (converting characters like `<` and `>` into their safe HTML entity representations), ensuring browsers parse inputs strictly as plain text rather than executable scripts.
- Implementing a Content Security Policy (CSP):** Configuring the web server to send strict HTTP headers that whitelist trusted script sources and block unauthorized inline scripts (`'unsafe-inline'`), preventing injected JavaScript from running.

Techniques to Prevent Race Conditions:

- Database Row-Level Locking:** Wrapping state-changing database operations in atomic transactions using locks (e.g. `SELECT FOR UPDATE` in SQL), ensuring a record cannot be read or altered by a competing thread until the current transaction commits.
- Atomic Operations and Mutexes:** In application code, wrapping access to shared memory variables in mutual exclusion locks (mutexes) ensures that only one thread can access and modify the shared resource at any given moment.

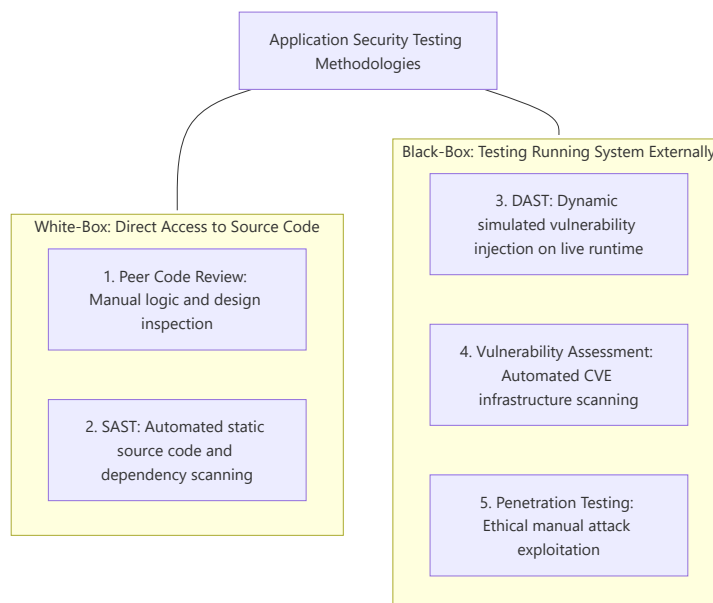
Chapter 11 - Security Verification, Hardening & Incident Response

Syllabus Dot Points Covered:

- *Developing secure code* — Test and evaluate the security and resilience of software by determining vulnerabilities, hardening systems, handling breaches, maintaining business continuity and conducting disaster recovery
- *Developing secure code* — Apply and evaluate strategies used by software developers to manage the security of programming code: code review, static application security testing (SAST), dynamic application security testing (DAST), vulnerability assessment, penetration testing

11.1 Security Verification: Code Analysis and Testing Strategies

Software engineers employ five complementary strategies across the development lifecycle to verify software resilience:



SECURITY TESTING STRATEGY	LIFECYCLE TIMING	SOURCE ACCESS REQUIRED?	OPERATIONAL MECHANICS, STRENGTHS & LIMITATIONS
Peer Code Review	Development / Sprints	Yes	Manual line-by-line inspection by developers. Catches complex architectural logic flaws and business-rule errors that automated scanners miss, but is time-consuming.
Static Application Security Testing (SAST)	Producing & Implementing	Yes	Automated tools (e.g. SonarQube) analyze raw source code, AST trees, and bytecode without running the application. Catches insecure functions early in CI/CD pipelines, but can produce false positives and misses runtime configuration errors.
Dynamic Application Security Testing (DAST)	Testing & Evaluating	No	Automated tools (e.g. OWASP ZAP) test the application from the outside while it is actively running, simulating attacks across HTTP/API endpoints. Finds real runtime and environmental configuration bugs, but cannot pinpoint the exact file line number of the flaw.

SECURITY TESTING STRATEGY	LIFECYCLE TIMING	SOURCE ACCESS REQUIRED?	OPERATIONAL MECHANICS, STRENGTHS & LIMITATIONS
Vulnerability Assessment	Testing & Deployment	No	Automated scans of servers, open ports, and operating system packages, cataloging discovered components against known Common Vulnerabilities and Exposures (CVE) databases. Provides broad coverage but does not demonstrate actual exploitability.
Penetration Testing	Pre-Production Evaluation	Optional (Black/Grey/White)	Authorized ethical hackers simulate adversarial attacks to actively chain and exploit weaknesses. Demonstrates real-world impact, but is expensive and relies heavily on tester skill.

Claritas Exam Tip: SAST vs DAST

In HSC exam questions (such as 2025 HSC Q13), remember:

- **SAST** runs during the **producing/implementing** stage and **requires** access to source code.
- **DAST** runs during the **testing/evaluating** stage on a compiled, executing system and **does not** require access to source code.

11.2 System Hardening

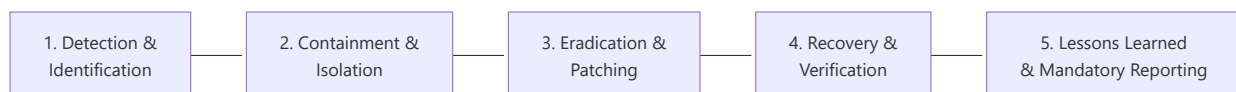
System hardening involves configuring systems to minimize their attack surface by removing non-essential software, services, and communication paths:

- **Least Functionality:** Disable and uninstall unnecessary operating system services, compilers, default sample files, and unused network daemons.

- **Port Closure:** Close all network ports except those strictly required for operation (e.g. closing FTP port 21 and Telnet port 23, exposing only HTTPS port 443).
 - **Elimination of Default Credentials:** Disable default vendor accounts (such as `admin/admin`), remove test user accounts, and enforce key-based SSH authentication with root login disabled.
 - **Principle of Least Privilege:** Execute application servers under unprivileged system service accounts rather than `root` or `Administrator`.
-

11.3 Incident Response: Handling Breaches

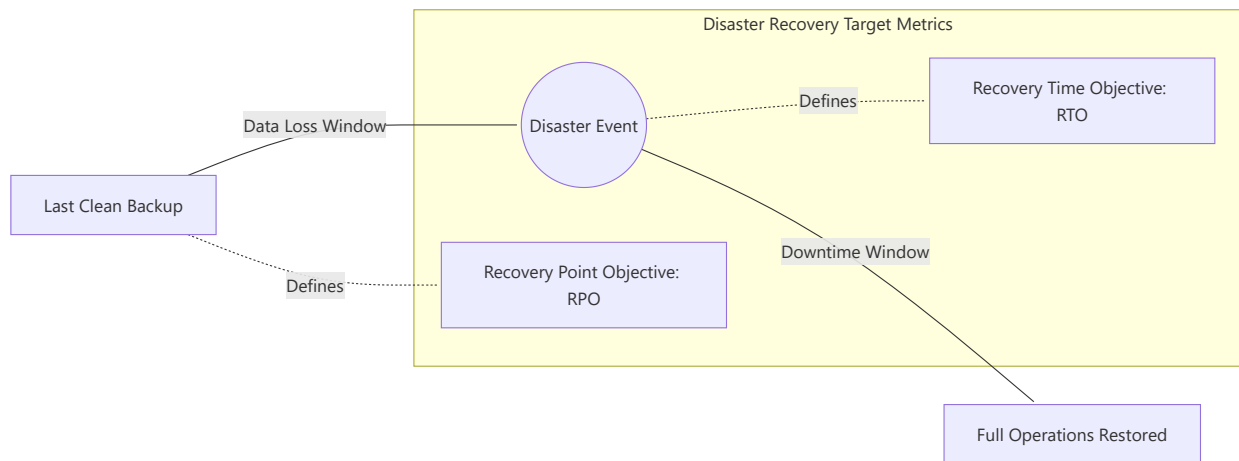
When a security breach occurs, the organization must execute a structured incident response plan to minimize damage, maintain evidence, and restore secure operations:



1. **Detection & Identification:** Intrusion Detection Systems (IDS) and log monitors flag unauthorized activity (e.g. an API endpoint returning high volumes of customer records to an unknown IP address).
 2. **Containment:** Stop the attack from spreading. Isolate affected network segments, revoke compromised credentials, and terminate hijacked sessions without wiping system memory.
 3. **Eradication:** Identify the root cause, remove attacker tools, close compromised entry points, and patch the exploited vulnerability.
 4. **Recovery:** Restore system operations from clean, verified backups, closely monitoring network traffic to ensure attackers have not left persistent backdoors.
 5. **Lessons Learned & Mandatory Reporting:** Conduct a post-mortem review to improve defensive controls. Under Australia's **Notifiable Data Breaches (NDB) scheme**, if personal information was exposed that is likely to result in serious harm, the organization is legally mandated to notify the Office of the Australian Information Commissioner (OAIC) and affected individuals within 30 days.
-

11.4 Business Continuity and Disaster Recovery

A resilient system must be able to withstand critical failures, power outages, and ransomware attacks.



- **Business Continuity Planning (BCP):** Strategies ensuring critical enterprise operations continue uninterrupted during an outage (e.g. using redundant cloud servers across multiple geographic regions with automated failover).
- **Disaster Recovery Planning (DRP):** Technical procedures for restoring systems and data after a catastrophic event.
- **Recovery Point Objective (RPO):** The maximum acceptable period of data loss measured in time. An RPO of 1 hour means the system must back up data at least hourly; up to 60 minutes of data loss is tolerable.
- **Recovery Time Objective (RTO):** The maximum acceptable duration of system downtime. An RTO of 4 hours mandates that engineers must have all core systems back online within 4 hours of an outage.

Chapter 11 Practice Questions

Multiple Choice Questions

Question 11.1 [Source: HSC Style Questions 2025, Q4]

Which of the following strategies involves executing an application to identify vulnerabilities in a running environment?

- (A) Code review
- (B) Static Application Security Testing (SAST)
- (C) Dynamic Application Security Testing (DAST)
- (D) Vulnerability assessment

Question 11.2 [Source: 2025 HSC Exam, Q13]

A software developer plans to use static application security testing (SAST) and dynamic application security testing (DAST) to evaluate an application. Which row correctly identifies the development timing and source code access required for SAST?

- (A) Testing and evaluating; Access required
- (B) Producing and implementing; Access required
- (C) Testing and evaluating; No access required
- (D) Producing and implementing; No access required

Short Answer Questions

Question 11.3 (3 marks) [Source: 2025 CSSA Trial, Q15b]

A café's Point of Sale (POS) system experiences unauthorized transaction deletions because all employee accounts share full administrative privileges. Explain how conducting a vulnerability assessment could help improve the security of this system.

Question 11.4 (4 marks) [Source: Claritas Original]

Distinguish between a Recovery Point Objective (RPO) and a Recovery Time Objective (RTO). In your response, explain how an enterprise's data backup schedule directly influences its achieved RPO.

Chapter 11 Marking Guidelines & Exemplars

Question 11.1 Solution

- **Correct Answer:** (C) Dynamic Application Security Testing (DAST).
- *Rationale:* DAST tools test applications from the outside while they are actively running, simulating attacks to identify runtime security flaws.

Question 11.2 Solution

- **Correct Answer:** (B) Producing and implementing; Access required.
- *Rationale:* SAST analyzes source code without running the application, so it requires source code access and is integrated into development and CI/CD pipelines during the producing and implementing stage.

Question 11.3 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly explains how a vulnerability assessment identifies security weaknesses in the POS system (e.g. lack of role-based access control, unauthenticated functions, insufficient logging). • Details the progression from risk identification to corrective remediation.	3
• Outlines how a vulnerability assessment identifies flaws with minor omissions.	2
• States a basic feature of vulnerability scanning.	1

Band 6 Sample Exemplar:

A vulnerability assessment would systematically evaluate the café's POS system to identify architectural and configuration weaknesses. It would highlight critical risks, such as the absence of

Role-Based Access Control (RBAC)—which allows regular staff to access sensitive management functions—and the lack of immutable audit logging for transaction deletions.

Based on these findings, the café can harden the POS system by restricting transaction deletions strictly to authenticated manager accounts and configuring automated audit trails that log employee IDs and timestamps for every transaction modification.

Question 11.4 Marking Guide

MARKING CRITERIA	MARKS
- Clearly defines both RPO (acceptable data loss window) and RTO (acceptable downtime window). - Accurately explains how backup frequency directly governs the RPO, showing that more frequent backups yield smaller RPOs.	4
Distinguishes between RPO and RTO with sound detail and outlines the connection to backups.	3
Outlines RPO and RTO with minor inaccuracies.	2
Identifies an isolated feature of backups or disaster recovery.	1

Band 6 Sample Exemplar:

Recovery Point Objective (RPO) defines the maximum acceptable volume of data loss measured in time between the last backup and a disaster event. **Recovery Time Objective (RTO)** defines the maximum acceptable duration of system downtime before core business operations must be restored.

An enterprise's data backup schedule directly determines its achieved RPO. If a financial database executes backups once every 24 hours (e.g. midnight), a catastrophic hardware failure occurring at 11:00 PM will result in up to 23 hours of unrecoverable transaction data, yielding an RPO of 24 hours. Conversely, implementing real-time database replication or continuous snapshots every 15 minutes achieves an RPO of 15 minutes, significantly minimizing data loss.

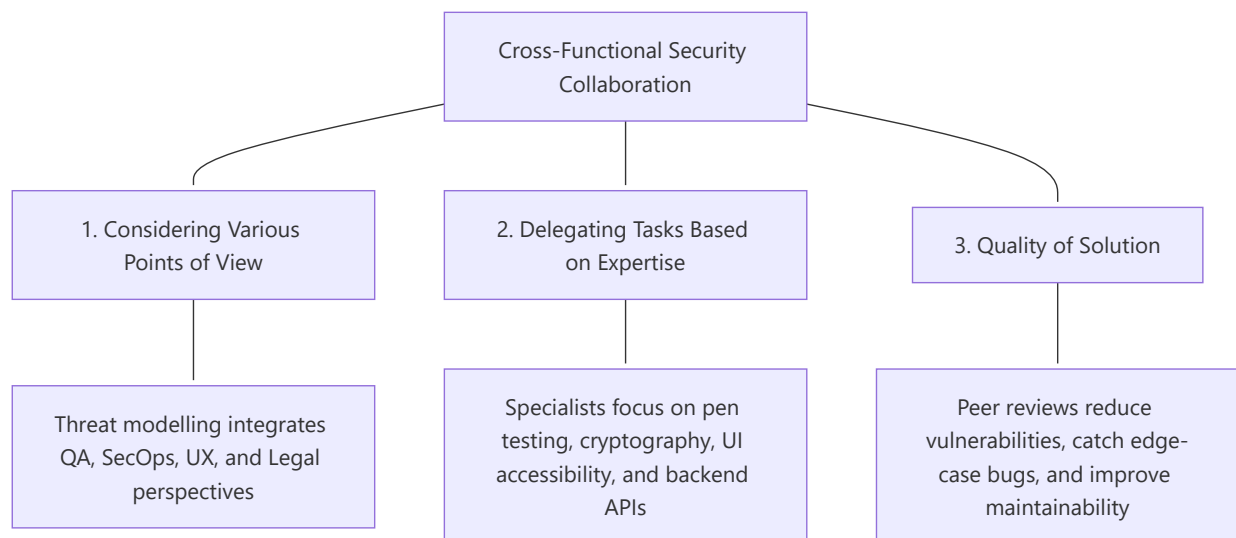
Chapter 12 - Enterprise Collaboration, Ethics, Law & Societal Impacts

Syllabus Dot Points Covered:

- *Impact of safe and secure software development* — Apply and describe the benefits of collaboration to develop safe and secure software: considering various points of view, delegating tasks based on expertise, quality of the solution
- *Impact of safe and secure software development* — Investigate and explain the benefits to an enterprise of the implementation of safe and secure development practices: improved products or services, influence on future software development, improved work practices, productivity, business interactivity
- *Impact of safe and secure software development* — Evaluate the social, ethical and legal issues and ramifications that affect people and enterprises resulting from the development and implementation of safe and secure software: employment, data security, privacy, copyright, intellectual property, digital disruption

12.1 Collaboration in Secure Software Engineering

Securing complex software systems exceeds the capacity of any single engineer. Modern software engineering relies on cross-functional collaboration to design, verify, and maintain secure code:



- **Considering Various Points of View:** Complex security weaknesses often hide at the intersections between system boundaries. Involving developers, UI/UX designers, database administrators, and legal compliance officers during initial threat modeling helps uncover edge cases that a single

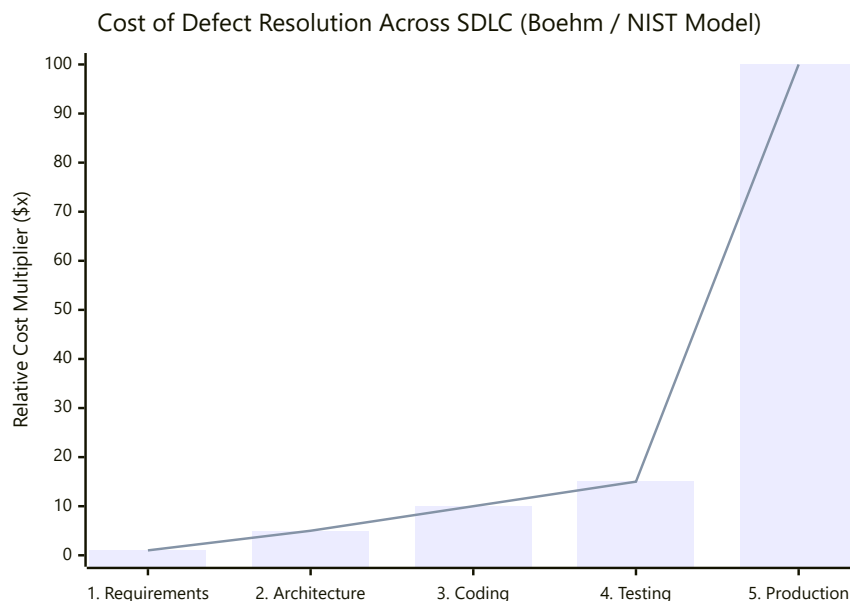
developer might overlook (e.g. a UX designer identifying that a complex password rule confuses elderly users, while a legal officer flags data residency requirements).

- **Delegating Tasks Based on Expertise:** Security requires specialized skills. Assigning cryptography configuration to security engineers, interface accessibility to front-end specialists, and transaction concurrency to database administrators ensures that critical components are built correctly according to current industry best practices.
 - **Quality of the Solution:** Practices like pair programming and peer code reviews establish shared ownership of the codebase. Multiple developers reviewing code reduces the likelihood of backdoors, forgotten debug credentials, or unvalidated input fields reaching production releases.
-

12.2 Enterprise Benefits of Secure Practices

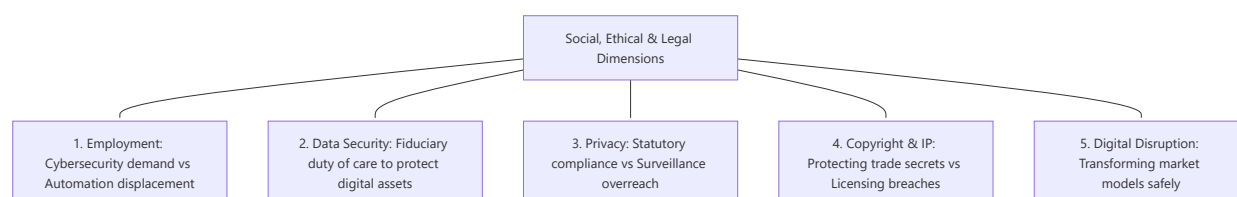
Investing in secure engineering practices delivers substantial, measurable returns across an organization:

1. **Improved Products and Services:** Software engineered with built-in security is more reliable, stable, and less prone to crashes, building user trust and strengthening brand reputation.
2. **Influence on Future Development:** Implementing secure coding standards establishes reusable templates, validated architectural patterns, and shared security libraries (e.g. a standardized company authentication module), accelerating the development of future projects.
3. **Improved Work Practices:** Integrating security checks directly into agile sprint pipelines normalizes continuous static testing and systematic code reviews, improving code consistency and professional discipline across the engineering team.
4. **Enhanced Productivity:** While secure development requires more upfront design effort, it saves significant time overall. Resolving an architectural vulnerability during the design phase is far less costly and time-consuming than fixing a production security breach under emergency conditions.
5. **Business Interactivity & Partner Trust:** Commercial enterprises must exchange data via APIs with banks, logistics partners, and government agencies. Demonstrating compliance with rigorous security standards (such as ISO/IEC 27001) enables organizations to establish trusted B2B integrations that expand market opportunities.



12.3 Evaluating Social, Ethical, and Legal Ramifications

Software security decisions have direct consequences for individual rights, corporate viability, and society as a whole:



1. Employment

- **The Talent Shift:** As organizations face increasing cyber threats, demand for skilled security engineers, ethical hackers, and security auditors continues to grow.
- **Ethical Obligation:** Automating administrative workflows with secure software can displace traditional clerical workers. Organizations have an ethical responsibility to provide transparent transition timelines and support staff reskilling programs.

2. Data Security

- **Legal Duty of Care:** Organizations hold a legal and ethical duty of care to protect the consumer data entrusted to them.
- **The Australian Legal Framework:** Under the **Privacy Amendment (Notifiable Data Breaches) Act 2017**, any organization subject to the Australian Privacy Principles (APPs) that experiences unauthorized access or disclosure of personal information must notify the affected individuals and the Australian Information Commissioner if the breach is likely to result in serious harm. Failure to implement reasonable security safeguards exposes organizations to statutory penalties exceeding \$50 million for serious or repeated privacy interferences.

3. Privacy

- **Privacy Rights vs Surveillance:** Engineers must balance application security logging with user privacy rights. While logging IP addresses and activity traces is essential for accountability, collecting excessive telemetry infringes on individual privacy. Systems should implement data minimization, collecting only the information strictly required for operation.

4. Copyright and Intellectual Property (IP)

- **Protecting Innovation:** Secure software architectures safeguard proprietary algorithms and trade secrets from corporate espionage.
- **Compliance with Open-Source Licenses:** Software teams must respect open-source copyright licenses. Inadvertently incorporating copyleft (e.g. GPL) code into proprietary commercial software can legally compel an enterprise to release its private source code publicly.

5. Digital Disruption

- **Market Evolution:** Secure software platforms enable transformative digital services, from app-based banking to telemedicine. However, these disruptions must be managed responsibly. If an entire economic sector becomes dependent on a single cloud service, a security breach or outage can disrupt national supply chains and essential public services.

Chapter 12 Practice Questions

Multiple Choice Questions

Question 12.1 [Source: 2025 NESA Sample Exam, Q4]

Which of the following are benefits of collaborating with others when developing secure software solutions? Select all that apply.

- ☐ Privacy is improved automatically without code
- ☐ Tasks can be delegated to those with specific expertise
- ☐ Different points of view and threat models can be considered
- ☐ The need for testing is eliminated
- ☐ Code is automatically exempt from open-source copyright

Question 12.2 [Source: 2025 CSSA Trial, Q9]

What is a key legal and ethical consideration related to the development of secure software?

- (A) Reducing the cost of development
- (B) Meeting data privacy regulations and compliance mandates
- (C) Increasing the visual complexity of the user interface
- (D) Writing software exclusively in a single programming language

Short Answer Questions

Question 12.3 (3 marks) [Source: Claritas Original]

Describe THREE benefits to an enterprise of embedding safe and secure development practices into its standard software engineering workflows.

Question 12.4 (8 marks) [Source: 2025 CSSA Trial, Q25 / Sample Trial 2025, Q23]

MetroWave, an Australian enterprise software provider, suffered an unauthorized data breach in its cloud-based Customer Relationship Management (CRM) system. Attackers exploited an authentication flaw in an exposed API, downloading confidential business reports, client correspondence, and proprietary product designs affecting over 200 corporate clients. The breach went undetected for two weeks.

Evaluate the social, ethical, and legal ramifications of this security incident for both MetroWave and its corporate clients.

Chapter 12 Marking Guidelines & Exemplars

Question 12.1 Solution

- **Correct Options:**

- ☒ ~~Tasks can be delegated to those with specific expertise~~
- ☒ ~~Different points of view and threat models can be considered~~

- *Rationale:* Collaboration allows teams to distribute tasks to domain specialists and identify a wider range of security risks during threat modeling. Collaboration does not remove the need for testing or alter legal copyright obligations.

Question 12.2 Solution

- **Correct Answer:** (B) Meeting data privacy regulations and compliance mandates.
- *Rationale:* Protecting user data and complying with statutory privacy laws (e.g. the Australian Privacy Act) is a fundamental legal and ethical duty in software development.

Question 12.3 Marking Guide

MARKING CRITERIA	MARKS
• Accurately describes THREE distinct enterprise benefits of secure software development practices (e.g. improved product quality, reduced long-term maintenance costs, partner trust/interactivity, templates for future projects).	3
• Outlines two enterprise benefits with sound detail.	2
• Identifies one enterprise benefit.	1

Band 6 Sample Exemplar:

1. **Improved Product Reliability and User Trust:** Software developed with security controls is less prone to crashes, logic bugs, and data breaches, which strengthens brand reputation and customer retention.
2. **Templates and Standards for Future Projects:** Establishing secure coding standards and libraries creates reusable modules (e.g. validated authentication routines) that accelerate development times on future enterprise projects.
3. **Reduced Incident Remediation Costs:** Fixing security vulnerabilities during early development saves significant time and money compared to the expense of emergency patching, legal counsel, and regulatory fines following a production breach.

Question 12.4 Marking Guide

MARKING CRITERIA	MARKS
- Comprehensively evaluates social, ethical, AND legal ramifications for BOTH MetroWave and its corporate clients. - Thoroughly discusses statutory compliance (Privacy Act / NDB scheme), breach of contract, professional negligence, reputation loss, and intellectual property exposure. - Integrates details from the scenario with sophisticated software engineering terminology.	7–8
Evaluates social, ethical, and legal issues for both parties with sound detail, though one area may lack depth.	5–6
Discusses some ramifications of the incident with limited evaluation or uneven coverage of the two parties.	3–4
Mentions basic consequences of a data breach.	1–2

Band 6 Sample Exemplar:

The data breach in MetroWave's CRM platform creates severe social, ethical, and legal ramifications across both organizations:

1. Social Ramifications:

- For MetroWave:* The company faces severe reputational damage. Trust is foundational for cloud software providers; losing client data will drive customer churn to competitors. Internally, employee morale will suffer, and engineers may face workplace stress or restructuring.
- For Corporate Clients:* Over 200 corporate clients face public embarrassment and loss of customer confidence. Leaked proprietary product designs compromise their commercial market advantage, potentially causing lost contracts and business disruption.

2. Ethical Ramifications:

- *For MetroWave:* Deploying an API with broken authentication represents a failure of professional engineering standards. Leaving the vulnerability unmonitored and undetected for two weeks demonstrates negligence in basic security monitoring. MetroWave breached its ethical duty of care to protect the confidential information entrusted to it.
- *For Corporate Clients:* Clients face the ethical challenge of transparently disclosing the breach to their own end-users whose personal data was exposed, balancing corporate reputation against their customers' right to know.

3. Legal Ramifications:

- *For MetroWave:* Under Australian law (**Privacy Act 1988** and the **Notifiable Data Breaches scheme**), MetroWave faces mandatory disclosure requirements to the OAIC. If found negligent in implementing reasonable security safeguards, MetroWave faces statutory regulatory fines of up to \$50 million. Additionally, corporate clients can pursue civil lawsuits for breach of contract and financial damages resulting from leaked trade secrets.
- *For Corporate Clients:* Clients whose customer data was compromised may themselves face regulatory scrutiny under privacy laws for failing to ensure third-party vendor compliance. They must also manage legal costs associated with investigating the breach and notifying affected individuals.

Conclusion:

This incident underscores that software security is not merely a technical concern, but a fundamental business and legal requirement. MetroWave must immediately patch its API authentication, conduct an external penetration test, cooperate with regulatory authorities, and implement rigorous continuous security testing across its development lifecycle.

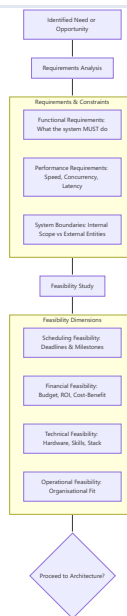
Chapter 13 - Project Inception, Feasibility & Development Approaches

Syllabus Dot Points Covered:

- *Identifying and defining* — Define and analyse the requirements of a problem: demonstrating need(s) or opportunities, assessing scheduling and financial feasibility, generating requirements including functionality and performance, defining boundaries
- *Identifying and defining* — Investigate types of software implementation methods: direct, phased, parallel, pilot
- *Research and planning* — Research and use the Waterfall software development approach: logical progression of steps used throughout the life cycle, stages of 'falling water', advantages and disadvantages, scale and types of developments
- *Research and planning* — Research and use the Agile software development approach: rate of developing a final solution, method tailoring, iteration workflow, scale and types of developments
- *Research and planning* — Research the WAgile software development approach: understanding it is a hybrid model, analysis of the 'when' and 'how' intervention is applied during the development life cycle, scale and types of developments
- *Research and planning* — Apply project management to plan and conduct the development and implementation of a project and software engineering solution: scheduling and tracking using a software tool, including Gantt charts, process diaries / log books

13.1 Defining Problems and System Feasibility

Every software engineering project originates from an identified business problem, an operational inefficiency, or an emerging commercial opportunity. Before writing code, software engineers must analyze requirements and establish project boundaries.



Functional vs Performance Requirements

- **Functional Requirements:** Define the specific operational behaviors, calculations, data transformations, and workflows the software must perform.
 - *Examples:* "The system must allow users to reset their password via an emailed token," or "The software must calculate tax withholding based on Australian ATO tax brackets."
- **Performance (Non-Functional) Requirements:** Impose verifiable constraints on the quality, efficiency, speed, or environmental conditions under which the system operates.
 - *Examples:* "The search API must return results within 250 milliseconds under a concurrent load of 10,000 active users," or "The database must support an uptime availability of 99.99%."

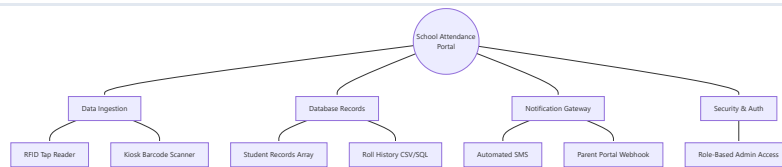
Defining System Boundaries

A **system boundary** defines what is inside the software system (the software's direct scope and automated processes) and what lies outside (external entities, users, third-party payment gateways, external database lookups). Clarifying boundaries prevents **scope creep**—the uncontrolled expansion of project scope without corresponding increases in time, budget, or resources.

Ideation Tools and Defining Data Requirements

During the inception phase, engineering teams deploy collaborative ideation tools to translate vague user needs into concrete requirements:

- **Brainstorming and Mind-Mapping:**
 - **Brainstorming:** An unconstrained, rapid-fire divergent ideation session designed to capture all possible feature pathways, user personas, and technical approaches without premature evaluation or criticism.
 - **Mind-Mapping:** A structured visual radial diagram originating from a central problem statement (e.g., "Automated School Attendance System"), branching into functional subsystems (hardware sensors, database storage, user notifications, parent portal). It establishes visual relationships, categorises scope, and exposes technical dependencies early.



- **Defining Data Requirements Early:**

Before database schemas are locked, engineers must catalog the **data types** and **data structures** necessary to satisfy the problem requirements:

1. **Identify Core Data Entities:** Determine what entities exist (e.g., `Student` , `ScanEvent` , `AbsenceReason`).
2. **Select Appropriate Data Types:** Map each attribute to its most memory-efficient type (e.g., 32-bit integer for `StudentID` , boolean for `IsExcused` , ISO 8601 string for `Timestamp`).
3. **Select Data Structures:** Determine whether data will reside in temporary in-memory arrays (for high-speed lookups during a school day) or append-only sequential disk files and relational tables (for end-of-term auditing).

Feasibility Study

Before capital is committed to development, a feasibility study assesses whether the project can be completed realistically:

1. **Scheduling Feasibility:** Assesses whether the system can be engineered, tested, and deployed within mandatory delivery timeframes (e.g., deploying an updated student reporting system before the start of the school term).
2. **Financial (Economic) Feasibility:** Evaluates development costs (labor, cloud infrastructure licenses, penetration testing) against anticipated commercial returns or cost reductions, using Cost-Benefit Analyses (CBA).
3. **Technical Feasibility:** Investigates whether the required technology, networking hardware, and developer programming skills exist within the organization or can be acquired.
4. **Operational Feasibility:** Determines whether the proposed software aligns with existing workplace practices, organizational culture, and legal compliance mandates.

13.2 Software Implementation (Conversion) Methods

When new software is ready to replace an existing operational system, organizations select an implementation method based on risk tolerance, organizational scale, and financial resources.

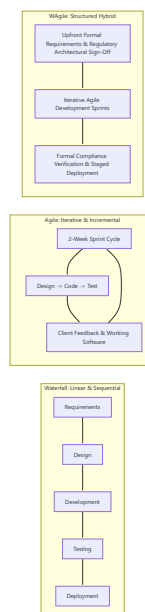


IMPLEMENTATION METHOD	OPERATIONAL MECHANICS	CORE ADVANTAGES	CORE DISADVANTAGES	IDEAL USE CASE SCENARIO
Direct Cutover	The old system is shut down completely and the new software starts operating immediately with no transition overlap.	• Lowest financial cost. • Minimal duration; no duplicate data entry. • Forces immediate user adoption.	• Highest operational risk. If the new software crashes, there is no backup system to fall back on, halting operations.	Non-critical systems, or situations where the old and new systems are fundamentally incompatible (e.g., a major database schema overhaul).
Parallel	Both the old and new systems run side-by-side simultaneously for a set period until the new system is fully verified.	• Lowest operational risk. Complete safety net; if the new system fails, business continues on the old platform.	• Expensive; double the hardware and hosting costs. • High employee workload due to entering every transaction twice.	Mission-critical systems where downtime or data corruption is unacceptable (e.g., banking cores, air traffic control).
Phased	The new software is introduced incrementally, module by module, gradually replacing older system components.	• Spreads financial and training load over time. • Isolates technical failures to a single subsystem without halting the entire enterprise.	• Can require temporary API bridges or middleware between old legacy modules and new modules. • Prolonged implementation timeline.	Large enterprise resource planning (ERP) suites (e.g., implementing the payroll module first, then invoicing six months later).

IMPLEMENTATION METHOD	OPERATIONAL MECHANICS	CORE ADVANTAGES	CORE DISADVANTAGES	IDEAL USE CASE SCENARIO
Pilot	The complete new system is implemented at one representative test site (or with a small cohort of users) before full deployment.	- Real-world feedback gathered from an active user group. - Operational bugs contained to the pilot site without affecting the wider business.	- Employees at the pilot site may experience initial disruptions. - Data synchronisation challenges between the pilot site and the rest of the business.	Multi-branch organizations (e.g., rolling out an ordering app to three test cafes before launching across a 200 store retail chain).

13.3 Project Methodologies: Waterfall, Agile, and WAgile

Choosing an appropriate software development life cycle (SDLC) model dictates how teams plan, code, verify, and adapt to changing requirements.



1. The Waterfall Approach

The **Waterfall model** is a sequential, plan-driven software engineering methodology. Development progresses steadily downwards through formal gates: Requirements → Design → Implementation → Verification → Maintenance.

- **Mechanics:** Each phase must be fully documented, reviewed, and signed off before the next phase begins. Feedback loops between non-adjacent phases are minimal.
- **Advantages:** Clear documentation, fixed budgets, predictable delivery timelines, and structured sign-off processes ideal for contractual arrangements.
- **Disadvantages:** Rigid and slow to adapt. Working software is not produced until late in the lifecycle; if initial requirements were flawed, discovering errors during testing requires expensive refactoring.
- **Scale & Suitability:** Large-scale, safety-critical, or defense engineering where regulatory compliance and architecture must be finalized upfront (e.g., avionics flight controllers, space launch control systems).

2. The Agile Approach

The **Agile model** prioritizes working software, customer collaboration, and responsiveness to change over comprehensive documentation and rigid plans.

- **Mechanics:** Development is partitioned into time-boxed iterations called **sprints** (typically 1–4 weeks). Each sprint follows a micro-lifecycle: planning, designing, coding, testing, and demonstrating working features to stakeholders.
- **Advantages:** Rapid deployment of working software, continuous user feedback, flexibility to pivot requirements mid-project, and early discovery of bugs.
- **Disadvantages:** Difficult to predict final project costs or completion dates; requires continuous client involvement; can suffer from documentation neglect.
- **Scale & Suitability:** Startups, commercial mobile applications, consumer web apps, and fast-moving markets where product requirements evolve rapidly based on user data.

3. The WAgile Approach (Hybrid)

WAgile (Waterfall-Agile) is a hybrid engineering approach designed to balance the regulatory structure of Waterfall with the iterative velocity of Agile.

- **Mechanics:** An initial Waterfall phase defines high-level system requirements, compliance criteria, and foundational database/security architectures. Once approved, the project transitions into an Agile model, where specific functional modules are developed and tested across iterative sprints. Finally, a structured Waterfall phase conducts system-level integration, compliance auditing, and staged deployment.
 - **Application of Intervention:** Formal stakeholder reviews occur upfront (approving specifications) and during final deployment gates, while developers use internal Agile stand-ups and sprint reviews during active feature construction.
 - **Scale & Suitability:** Heavily regulated industries (banking, medical systems, public sector platforms) that must prove compliance with statutory laws while still benefiting from modern iterative engineering practices.
-

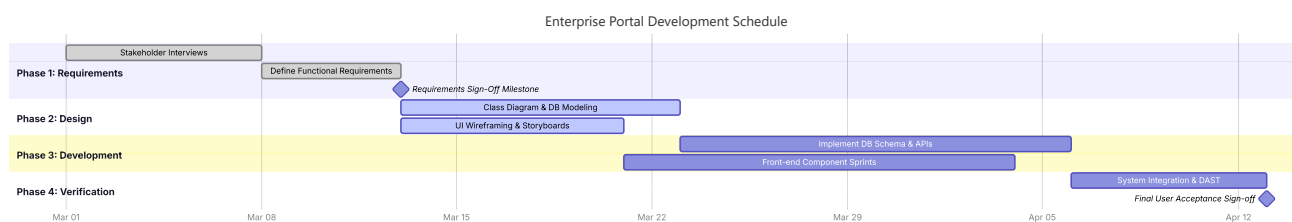
13.4 Project Management Tools: Tracking & Collaboration

Delivering engineering projects on schedule requires tools that visualize task dependencies and record technical decisions.

Gantt Charts

A **Gantt chart** is a project management tool that maps component tasks against a chronological calendar timeline. It visualizes:

- **Task Durations:** Represented by horizontal bars whose length corresponds to estimated effort.
- **Dependencies:** Arrows connecting tasks, showing that dependent tasks cannot start until predecessor tasks complete (e.g., "Database Implementation" cannot begin until "Entity-Relationship Design" is approved).
- **Milestones:** Critical decision points, deliverables, or phase completions, typically represented by diamond symbols (◊).



Process Diaries and Log Books

According to the **NESA Course Specifications**, software engineers must maintain process diaries/log books to document technical progress. Entries must be recorded at regular intervals and include:

- Date and person making the entry.
- Specific progress achieved since the last entry.
- Stumbling blocks or technical bottlenecks encountered, and how they were managed.
- Planned approaches for upcoming tasks.
- Reflective self-evaluation comments.
- Hardware, software, and intellectual resources utilized.

Software Collaboration Tracking Tools

Modern engineering relies on digital project boards (e.g., Jira, Trello, GitHub Projects) implementing the **Kanban workflow**:

- **Backlog:** Unprioritised pool of user stories and feature requests.
- **In Progress (WIP):** Active tasks assigned to specific engineers, with explicit Work-In-Progress limits to prevent multitasking bottlenecks.
- **Peer Review / QA:** Code submitted via pull requests undergoing testing and review before sign-off.
- **Done:** Completed, validated code deployed to the build target.

Chapter 13 Practice Questions

Multiple Choice Questions

Question 13.1 [Source: 2025 HSC Exam, Q1]

A streaming platform updated its content suggestion algorithm for users. A select group of users was upgraded to the new algorithm, and feedback was collected before replacing the algorithm for all users. Which implementation method was used?

- (A) Pilot
- (B) Direct
- (C) Phased
- (D) Parallel

Question 13.2 [Source: 2025 HSC Exam, Q2]

Complete the statement by choosing the correct terms:

The ____ approach typically requires extensive documentation, while the ____ approach often emphasizes working software over documentation.

- (A) Agile; Waterfall
- (B) Waterfall; Agile
- (C) Phased; Direct
- (D) Parallel; WAgile

Short Answer Questions

Question 13.3 (4 marks) [Source: HSC Style Questions 2025, Q16 / Sample Trial 2025, Q22b]

Compare and contrast the Waterfall and WAgile (Hybrid) software development methodologies in the context of developing control software for an industrial toy manufacturing company that must adhere to strict government safety regulations.

Question 13.4 (3 marks) [Source: Claritas Original]

An accounting firm is updating its client invoicing software. Explain ONE operational advantage and ONE operational disadvantage of selecting a **parallel implementation method** over a **direct cutover**.

Chapter 13 Marking Guidelines & Exemplars

Question 13.1 Solution

- **Correct Answer:** (A) Pilot.
- *Rationale:* Trialling the complete new algorithm on a small representative subgroup of users to collect feedback before rolling it out organization-wide is the definition of a pilot implementation.

Question 13.2 Solution

- **Correct Answer:** (B) Waterfall; Agile.

- *Rationale:* The Waterfall model requires extensive documentation at each stage before progressing, whereas the Agile Manifesto explicitly prioritizes working software over comprehensive documentation.

Question 13.3 Marking Guide

MARKING CRITERIA	MARKS
• Thoroughly compares AND contrasts Waterfall and WAgile methodologies within the context of regulated manufacturing software. • Explains structural similarities (both require upfront compliance documentation and testing gates). • Explains key differences (Waterfall executes linearly with late software delivery; WAgile uses iterative coding sprints between formal gates).	4
• Compares Waterfall and WAgile with sound detail, referencing regulatory or manufacturing aspects.	3
• Outlines characteristics of Waterfall and/or WAgile with limited contrast.	2
• States basic definitions of Waterfall or Agile.	1

Band 6 Sample Exemplar:

In developing software for safety-regulated toy manufacturing, both Waterfall and WAgile share the need for rigorous upfront planning to define safety specifications, hardware constraints, and compliance mandates enforced by regulatory agencies.

However, they contrast fundamentally in execution:

Under a **Waterfall** approach, development proceeds sequentially: all requirements, architectural blueprints, and compliance documents must be finalized before coding begins. The control software is delivered and tested all at once at the end of the lifecycle. If machine sensor integrations or safety interlocks fail during testing, fixing them requires expensive rework across the entire documentation chain.

In contrast, **WAgile (Hybrid)** uses Waterfall's formal documentation phase upfront to satisfy mandatory safety certifications, but executes the active development phase through iterative **Agile sprints**. Developers iteratively code, test, and refine specific machine controller modules (e.g., conveyor belts, robotic cutters) with immediate feedback from physical assembly hardware. This allows the team to identify hardware-software integration bugs early while preserving the formal verification gates required for regulatory sign-off.

Question 13.4 Marking Guide

MARKING CRITERIA	MARKS
- Explains ONE clear operational advantage of parallel implementation within an accounting context (e.g., zero downtime risk, verifiable financial auditing). - Explains ONE clear operational disadvantage (e.g., double data entry workload, higher infrastructure costs).	3
Outlines one advantage and one disadvantage with minor omissions.	2
States an isolated feature of parallel implementation.	1

Band 6 Sample Exemplar:

Operational Advantage: Parallel conversion provides a complete safety net that minimizes risk for the accounting firm. Invoicing transactions are entered into both the old and new systems simultaneously, allowing accountants to cross-check calculated invoice totals and ledger entries. If the new software encounters unexpected bugs or crashes, client billing continues uninterrupted on the old platform without financial loss.

Operational Disadvantage: Parallel conversion places a heavy operational workload on employees, as accounting staff must enter every invoice, payment, and client detail twice. This redundant data entry increases operational labor costs and can lead to staff fatigue and manual data entry errors.

Chapter 14 - System and Data Modelling Tools

Syllabus Dot Points Covered:

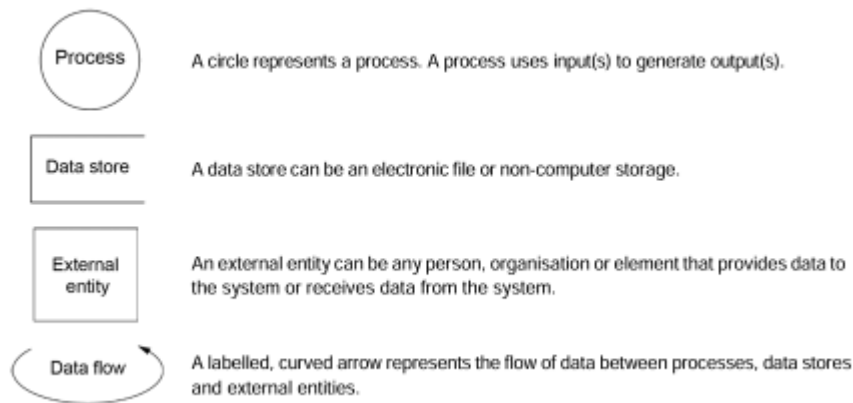
- *Research and planning* — Demonstrate the use of modelling tools
- *Course Specifications* — Data flow diagrams: symbols, Level 0 (Context) vs Level 1 DFDs
- *Course Specifications* — Structure charts: symbols, data movement, control flags, decisions, repetition, refinement
- *Course Specifications* — Data dictionary: variable name, data type, display format, size in bytes, display size, description, example, validation
- *Course Specifications* — Class diagrams: class name, attributes, methods, inheritance, relationships, multiplicity
- *Course Specifications* — Storyboards: interface layouts, navigation paths
- *Course Specifications* — Decision trees: conditions, branching, outcomes

14.1 Data Flow Diagrams (DFDs)

Data Flow Diagrams (DFDs) trace how data moves through an information system, detailing inputs, storage locations, transformations, and outputs, without showing programming control structures (like loops or if-statements).

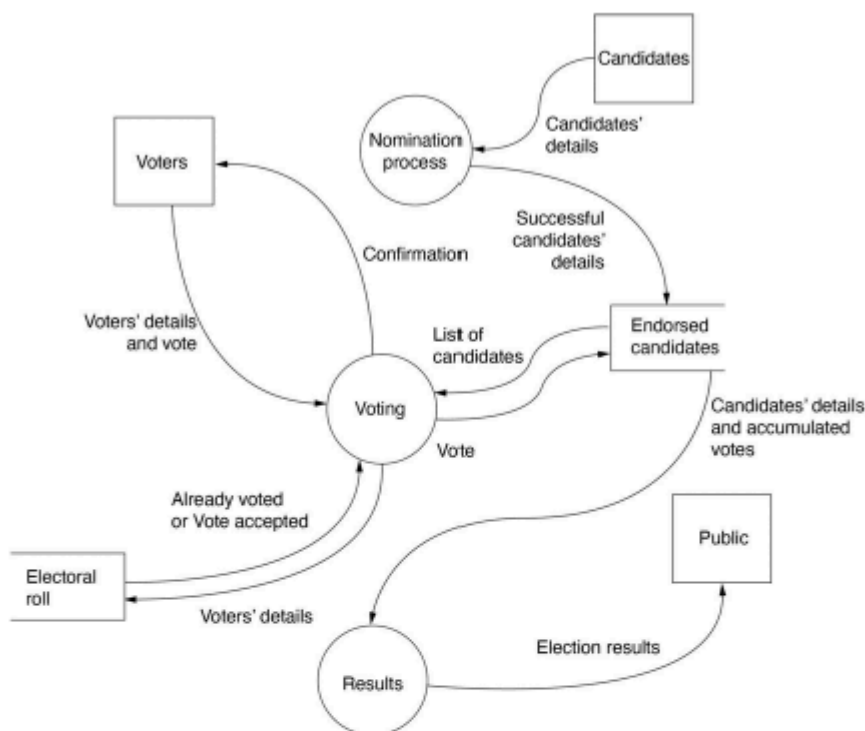
DFD Notation Standards (NESA Course Specifications)

- **External Entity (Rectangle):** A person, organization, or external system that provides data to or receives data from the system.
- **Process (Circle):** A transformation step that takes incoming data and transforms it into outgoing data. Must be named using a verb-noun phrase (e.g., `Verify Voter Details`).
- **Data Store (Open-ended Rectangle):** A repository of persistent data (database table, file, manual filing cabinet).
- **Data Flow (Curved Arrow):** A labelled arrow showing the movement of discrete data items between entities, processes, and data stores.



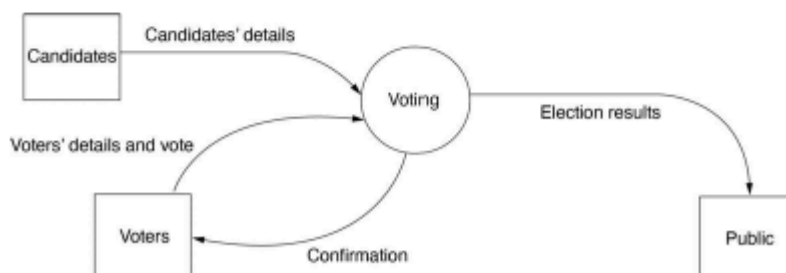
Level 0 (Context) vs Level 1 DFDs

- **Level 0 (Context Diagram):** Depicts the system as a single process circle connected to its external entities. It shows the system's boundary and inputs/outputs, but hides internal processes and data stores.
- **Level 1 DFD:** Decomposes the single system process into its major functional subprocesses, revealing internal data stores and intermediate data flows.



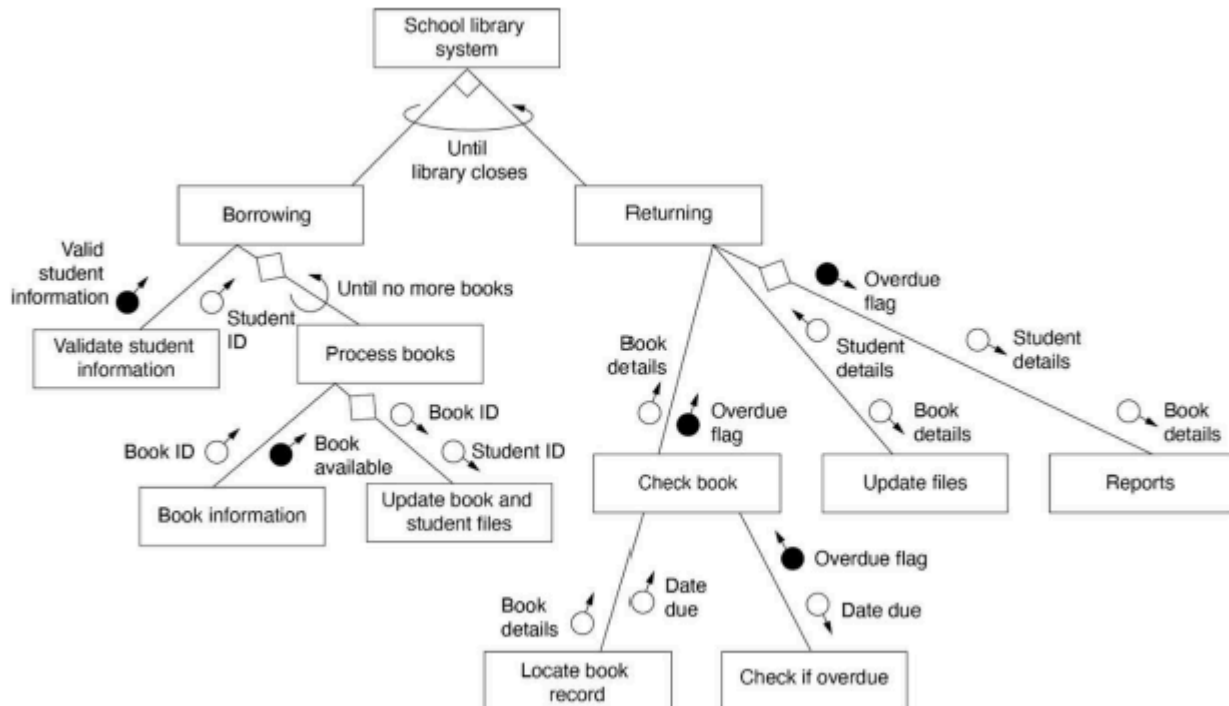
Level 0 data flow diagram

Level 0 data flow diagrams represent an overview of the entire system and do not show data stores or internal processes. The following represents a Level 0 data flow diagram for the voting system.



14.2 Structure Charts

While DFDs show how data flows through a system, **Structure Charts** represent the system's code architecture. They show the top-down modular decomposition of a program into subroutines and depict the parameters and control variables exchanged between them.



Notation Standards (NESA Course Specifications)

- **Subroutine Box:** A rectangle representing a distinct module, function, or procedure.
- **Data Movement Couple (Empty Circle with Arrow ○→):** Indicates a data variable passed as a parameter between modules.
- **Control Flag Couple (Filled Circle with Arrow ●→):** Indicates a status flag or control variable passed between subroutines to alter execution logic (e.g., `Valid`, `EOF`, `OverdueFlag`).
- **Conditional Decision (Diamond ◇):** Placed at the intersection of calling lines to indicate that calling a subordinate subroutine is optional, depending on a selection condition.
- **Repetition (Curved Arrow ↺):** Placed across connection lines to indicate that a subordinate subroutine is called repeatedly in a loop.
- **Refinement:** Decomposing complex modules into smaller submodules across multiple levels of abstraction.



An empty circle is used to indicate data movement between subroutines (usually passed as parameters).



A filled circle is used to indicate a flag or control variable that is passed between subroutines.



A decision (ie optional execution of a subroutine) is indicated by use of a small diamond at the intersection of the connecting lines between subroutines that are called as the result of a binary or multi-way selection. Alternatively, the diamond may appear on a single connecting line if calling that subroutine is optional. In the diagram shown, a report may not necessarily be produced each time a book is returned.



Repetition (execution of a particular subroutine or set of subroutines multiple times) is shown by a curved arrow.

14.3 Data Dictionaries

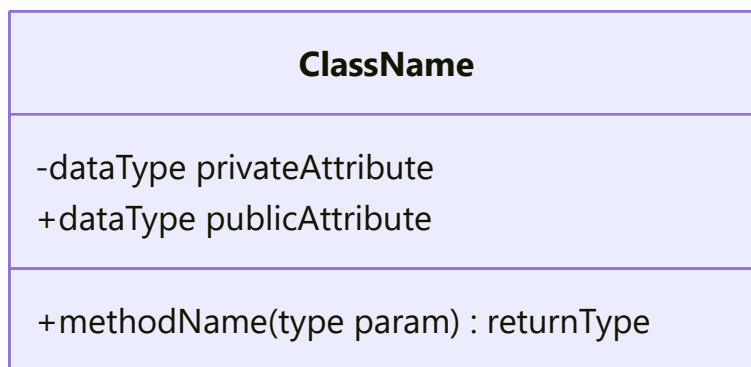
A **Data Dictionary** provides a structured catalog of every variable, field, and data structure in an application, defining its data type, memory footprint, display formatting, and validation constraints.

VARIABLE NAME	DATA TYPE	FORMAT FOR DISPLAY	SIZE IN BYTES	DISPLAY SIZE	DESCRIPTION & PURPOSE	VALIDATION
MemberID	String	XNNNN	5 bytes	5 chars	Unique primary identifier for patron account.	Starts with 'Z', followed by 4 digits
FullName	String	XX...XX	30 bytes	30 chars	Full legal name of library patron.	String between 30 characters
BooksLoaned	Integer	NN	2 bytes	2 digits	Number of books currently on loan to member.	Integer ≥ 0
FineBalance	Real / Float	\$NNN.NN	4 bytes	7 chars	Monetary value of outstanding unpaid late fines.	Number ≥ 0.00

VARIABLE NAME	DATA TYPE	FORMAT FOR DISPLAY	SIZE IN BYTES	DISPLAY SIZE	DESCRIPTION & PURPOSE	VALID
HasSuspension	Boolean	X	1 bit	1 char	Flag indicating whether borrowing privileges are frozen.	Mus stric or F
DateOfBirth	Date / Timestamp	YYYY-MM-DD	4 bytes	10 chars	Stored as a 32-bit epoch integer representing birthdate.	Date chr earli toda

14.4 Class Diagrams

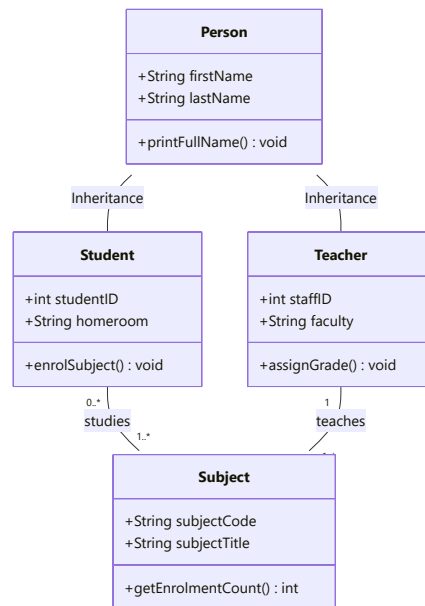
Class diagrams model the object-oriented structure of a software system, mapping classes, their attributes and methods, and the relationships between them.



Notation Standards (NESA Course Specifications)

- **Inheritance (→ with closed hollow triangle):** Indicates an *is-a* relationship where a specialized child subclass inherits attributes and methods from a generalized parent base class.
- **Association / Relationship (→ with open arrowhead):** Indicates a structural relationship where one class uses or references another.
- **Multiplicity Indicators:**
 - 1..1 (or 1): Exactly one instance.
 - 0..1: Zero or one instance (optional).
 - 1..*: One or more instances.
 - 0..*: Zero or more instances (many).

- **m..n** : At least m and at most n instances.

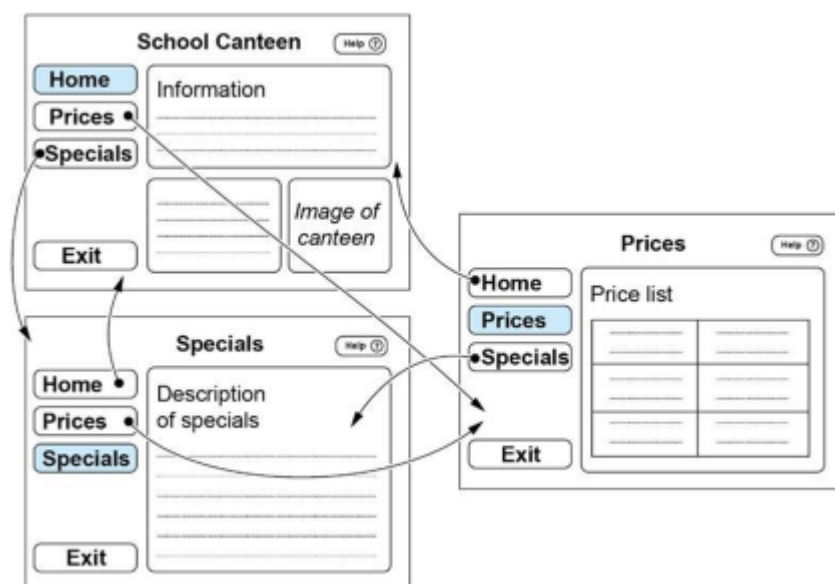


14.5 Storyboards and Decision Trees

Storyboards

A **storyboard** visually maps the user interface, detailing screen layouts and the navigation links between them. According to the Course Specifications, storyboards document:

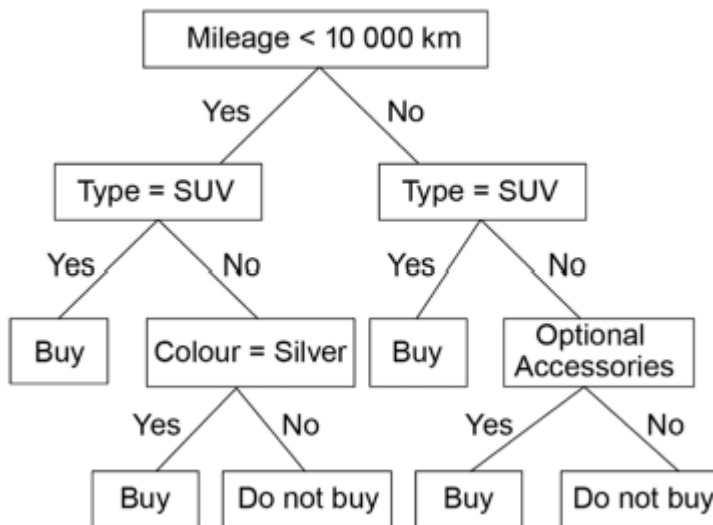
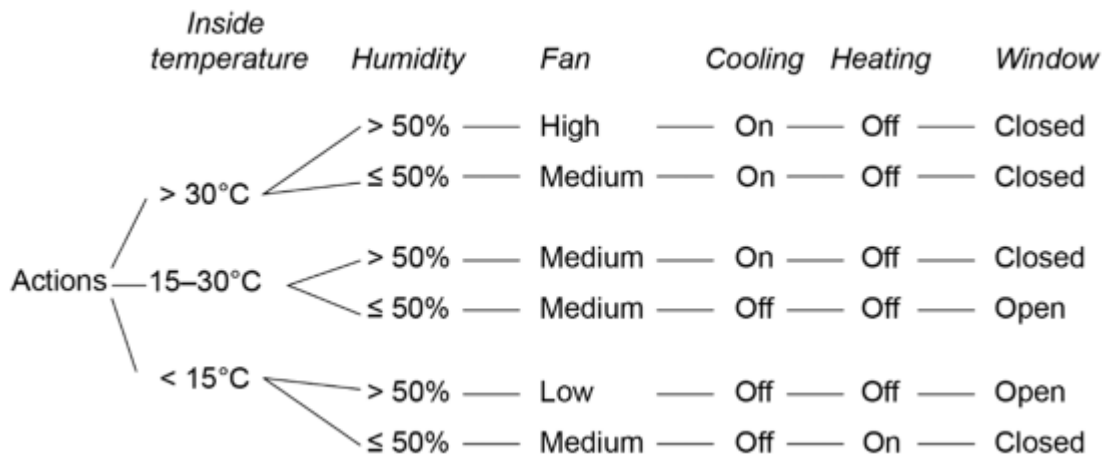
- Placement of UI elements: headers, form inputs, buttons, media panes.
- Navigation paths connecting pages (e.g., links from a Home page to a Checkout or Search page).
- Dynamic interface states (e.g., showing pop-up modals or error message placements).



Decision Trees

A **decision tree** provides a complete graphical map of complex conditional logic, showing every path of decisions and their resulting actions.

- **Nodes:** Represent conditions evaluated on specific feature values.
- **Branches:** Represent the outcomes of those conditions (Yes/No, True/False, or categorical values).
- **Terminal Actions:** The final process, state, or prediction reached along that decision path.



Chapter 14 Practice Questions

Multiple Choice Questions

Question 14.1 [Source: 2025 HSC Exam, Q22]

A software system contains an `Animal` base class with subclasses `Mammal` and `Reptile`. Subclasses `Whale` and `Dog` inherit from `Mammal`. Which object-oriented relationship is used to remove redundant attributes like `Colour` and `Size` from the child classes?

- (A) Multi-way selection
(B) Inheritance

- (C) Encapsulation
- (D) Polymorphic dynamic dispatch

Question 14.2 [Source: 2025 CSSA Trial, Q22a]

Which of the following modelling tools is designed specifically to represent the functional subroutines of an application and the parameters passed between them?

- (A) Level 0 Data Flow Diagram
- (B) Gantt Chart
- (C) Structure Chart
- (D) Storyboard

Short Answer Questions

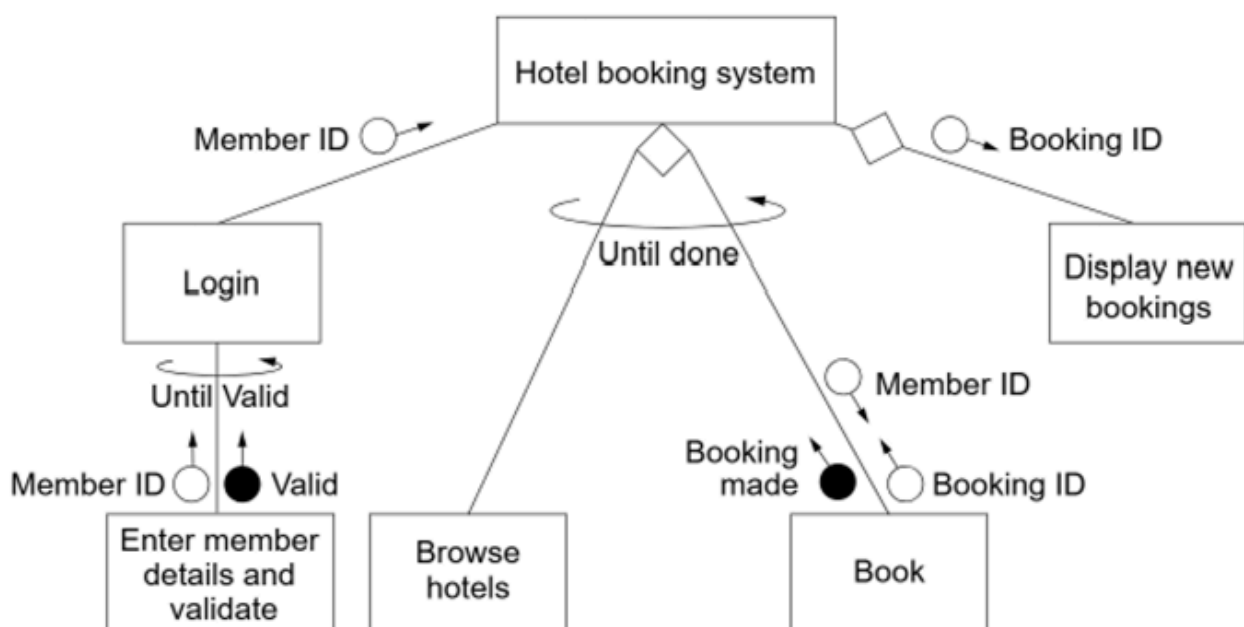
Question 14.3 (4 marks) [Source: 2025 HSC Exam, Q22 / NESA Course Specs, p. 9]

An airline booking system manages long-distance passenger flights. The system requires modeling `TrainTrip`, `Engine`, `PassengerCar`, `Area` (Standard and Executive), and `Passengers`. Construct a UML class diagram that illustrates classes, attributes, methods, inheritance, and multiplicity relationships for this system.

Question 14.4 (3 marks) [Source: 2025 NESA Sample Exam, Q10 / HSC Familiarisation Questions, Q4]

Examine the following structure chart representing an online ordering system:

This structure chart describes a hotel booking system.



Identify the graphical symbol used to depict a **control flag**, describe how it differs from a **data movement couple**, and give an example of when a control flag would be returned by `Validate User`.

Chapter 14 Marking Guidelines & Exemplars

Question 14.1 Solution

- **Correct Answer:** (B) Inheritance.
- *Rationale:* Inheritance allows child classes to automatically derive shared attributes and methods from a common parent class, eliminating redundant field definitions across subclasses.

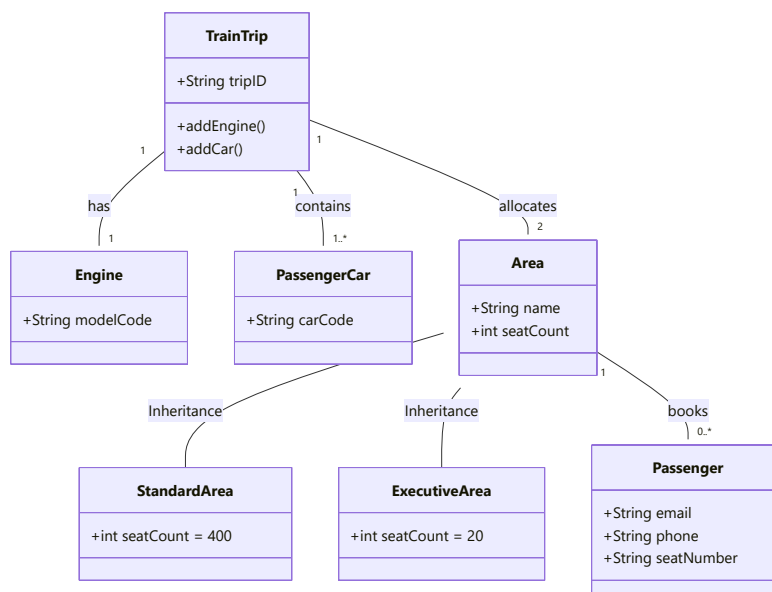
Question 14.2 Solution

- **Correct Answer:** (C) Structure Chart.
- *Rationale:* Structure charts model software modularity, showing subroutines, parameter passing (data couples), and control flags.

Question 14.3 Marking Guide

MARKING CRITERIA	MARKS
• Accurately draws a UML class diagram incorporating suitable class boxes with attributes and methods. • Illustrates correct inheritance (e.g., Standard and Executive inheriting from an Area base class). • Depicts relationships with correct multiplicity indicators (1..1, 1..*).	4
• Draws a class diagram representing most classes, relationships, and multiplicity.	3
• Draws a class diagram with partial attributes, classes, and relationships.	2
• Shows basic understanding of class diagram notation.	1

Band 6 Sample Exemplar:



Question 14.4 Marking Guide

MARKING CRITERIA	MARKS
- Identifies the control flag symbol (solid/filled circle with an arrow). - Clearly explains the difference between a control flag and a data couple. - Provides a realistic example for the scenario (e.g. <code>IsValidUser</code> boolean flag).	3
Outlines the control flag symbol and distinguishes it from a data couple with minor omissions.	2
States the symbol or identifies a control flag example.	1

Band 6 Sample Exemplar:

A control flag is represented by a **filled (solid) circle with a directed arrow** ($\bullet \rightarrow$). It differs from a data movement couple (which uses an empty/hollow circle $\circ \rightarrow$ to carry raw data values like a username string) because a control flag carries a directive or status condition used to alter execution flow in the calling module.

In this scenario, `Validate User` would evaluate the entered credentials and return a Boolean control flag such as `IsValid` (`True` or `False`) to the main controller, determining whether execution proceeds to `Process Cart` or halts with an error.

Chapter 15 - Algorithm Design, Data Structures & Implementation

Syllabus Dot Points Covered:

- *Producing and implementing* — Design, construct and implement a solution to a software problem using appropriate development approach(es)
 - *Producing and implementing* — Develop, construct and document algorithms: pseudocode, flowcharts, subprograms, parameter passing
 - *Course Specifications* — Control structures: Sequence, Binary selection, Multi-way selection (`CASEWHERE`), Nested IF, Repetition (Pre-test, Post-test, Counted loops)
 - *Course Specifications* — Subroutines: parameters, functions returning values
 - *Testing and evaluating* — Use a language-dependent code optimisation technique
-

15.1 NESA Pseudocode & Flowchart Standards

Algorithms must be expressed unambiguously using standardized control structures, keywords, and indentation conventions.

Pseudocode Standards (NESA Course Specifications)

1. **Capitalized Keywords:** Keywords indicating control structures and operations must be capitalized (e.g., `BEGIN` , `END` , `IF` , `THEN` , `ELSE` , `ELSEIF` , `ENDIF` , `WHILE` , `ENDWHILE` , `REPEAT` , `UNTIL` , `FOR` , `TO` , `STEP` , `NEXT` , `CASEWHERE` , `ENDCASE` , `RETURN`).
2. **Paired Structural Elements:** Control structures must be balanced with matching termination statements (`IF ... ENDIF` , `WHILE ... ENDWHILE`).
3. **Indentation:** Used consistently to visually indicate nesting within control structures.
4. **Modularity:** Mainline algorithms reference subroutines by name, with subroutine logic defined in a separate named block.

```
// =====  
// NESA Standard Pseudocode Template: Modular Function Pattern  
// =====  
BEGIN Mainline  
    Get input_value  
    calculated_result = CalculateDiscount(input_value)  
    DISPLAY "Discounted total: $", calculated_result  
END Mainline  
  
BEGIN CalculateDiscount (base_amount)  
    IF base_amount > 100 THEN  
        discount_rate = 0.20  
    ELSEIF base_amount >= 50 THEN  
        discount_rate = 0.10  
    ELSE  
        discount_rate = 0.00  
    ENDIF  
  
    final_amount = base_amount * (1.0 - discount_rate)  
    RETURN final_amount  
END CalculateDiscount
```

Flowchart Standards

Flowcharts are read from top to bottom and left to right, connecting standard geometric symbols with directional arrowheads:

- **Terminator (Rounded Oval):** Indicates the algorithm start, end, or subroutine entry/exit (**BEGIN** / **END**).
- **Process (Rectangle):** Represents a data manipulation or assignment operation.
- **Input/Output (Parallelogram):** Indicates reading external data or displaying output.
- **Decision (Diamond):** Evaluates a conditional expression, with output branches explicitly labelled (**True** / **False** or **Yes** / **No**).
- **Subprogram / Predefined Process (Double-Strikethrough Rectangle):** Calls an externally defined subroutine.



15.2 Control Structures Reference

CONTROL STRUCTURE	NESA PSEUDOCODE SYNTAX	OPERATIONAL LOGIC & LOOP CONDITION
Sequence	<pre> process 1 process 2 </pre>	Executes sequentially in exact order of appearance without branching.
Binary Selection	<pre> IF condition THEN process 1 ELSE process 2 ENDIF </pre>	Evaluates condition: executes branch 1 if True, optional branch 2 if False.
Multi-way Selection (Case Structure)	<pre> CASEWHERE expression evaluates to choice a : process a choice b : process b OTHERWISE : default process ENDCASE </pre>	Evaluates expression once and matches against discrete choices. Ceases evaluation once a path executes.
Pre-Test Loop (Guarded Loop)	<pre> WHILE condition is true process ENDWHILE </pre>	Tests condition BEFORE the loop body. Body may execute zero times if condition starts False.
Post-Test Loop	<pre> REPEAT process UNTIL condition is true </pre>	Tests condition AFTER loop body. Loop body is GUARANTEED to execute at least once.
Counted Loop	<pre> FOR var = start TO finish STEP inc statements NEXT var </pre>	Special repetition where an index variable increments from start to finish values across fixed steps.

15.3 Data Structures: Implementation in Algorithms

Selecting appropriate data structures is fundamental to writing efficient software:

1. Single and Multidimensional Arrays

- **1D Arrays:** Contiguous sequences of elements indexed numerically (e.g., `Risks[0]` through `Risks[9]`).
- **2D Arrays (Matrices):** Tabular data structures indexed via rows and columns (`Grid[row][col]`).
- *Course Specs Seating Plan Pattern:* Managing seat reservations across an aircraft cabin:

Business Class

	1	2	3	4	5	6	7	8
A	X	X	X	X				
B	X	X	X	X	X			
C	X	X	X	X	X	X	X	
D								
E								

```
# Initialising a 2D Array: 5 rows by 8 columns (Business Class Seating)
rows = 5
cols = 8
seating_plan = [["Available" for _ in range(cols)] for _ in range(rows)]

def reserve_adjacent_seats(grid, num_tickets):
    """Searches 2D array for contiguous available seats in a single row."""
    for r in range(len(grid)):
        for c in range(len(grid[0]) - num_tickets + 1):
            # Check if required adjacent seats are unallocated
            if all(grid[r][c + k] == "Available" for k in range(num_tickets)):
                for k in range(num_tickets):
                    grid[r][c + k] = "Reserved"
                return [(r, c + k) for k in range(num_tickets)]
    return None # Allocation failed
```

2. Sequential Files

Files stored on secondary disk storage parsed sequentially from beginning to end:

- Operations follow strict sequence: `OPEN` , read line by line using an `EOF` (End of File) pre-test while loop, `CLOSE` .

15.4 Language-Dependent Code Optimisation

Code optimization involves refining algorithms to consume fewer CPU clock cycles, utilize less memory, or take advantage of high-performance built-in language routines.

Python Code Optimization Examples

- **Optimisation 1: Built-in Functions vs Manual Loops:**

```
# Unoptimised: Interpreted Python loop with manual accumulation
numbers = [1, 2, 3, 4, 5]
total = 0
for num in numbers:
    total += num

# Optimised: Language-dependent C-optimised native built-in
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)
```

Why it is faster: The native `sum()` function is implemented in optimized C within the CPython runtime, avoiding interpreter bytecode dispatch overhead on each iteration.

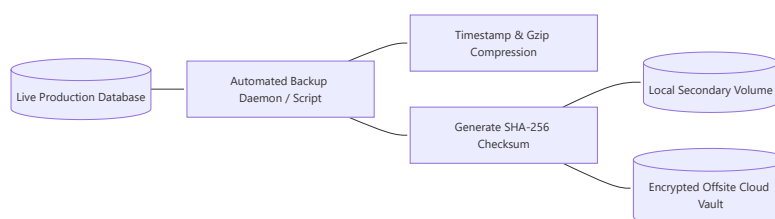
- **Optimisation 2: List Comprehensions vs Append Loops:**

```
# Unoptimised: Dynamic array resizing and repeated function attribute lookups
squares = []
for x in range(1000):
    squares.append(x ** 2)

# Optimised: Memory pre-allocation using list comprehension
squares = [x ** 2 for x in range(1000)]
```

15.5 Programmed Data Backup and Automated Archival

Data is the most valuable asset in any software engineering ecosystem. Operating systems crash, hardware disks fail, and ransomware attacks corrupt file trees. Meeting enterprise resilience standards requires implementing **programmed data backups**—automated, scheduled routines that archive application databases and user assets without human intervention.



The 3-2-1 Backup Strategy

A production software solution must implement the industry-standard **3-2-1 backup principle**:

1. Maintain at least **three (3)** distinct copies of application data.

2. Store the data across at least **two (2)** different physical media types (e.g., fast NVMe SSD server disk + cloud object storage).
3. Keep at least **one (1)** copy in an **offsite, geographically isolated** cloud location to survive physical facility disasters.

Python Programmed Backup Automation Script

In software projects, automated backup scripts execute via operating system schedulers (such as `cron` jobs in Linux or Task Scheduler in Windows). The following production-ready script demonstrates programmed directory traversal, timestamped archive generation, and cryptographic hashing:

```

# =====
# Programmed Data Backup and Verification Script
# Claritas Software Engineering Reference Suite
# =====

import os
import shutil
import hashlib
from datetime import datetime

SOURCE_DATA_DIR = "./production_database"
BACKUP_VAULT_DIR = "./backup_vault"

def generate_file_sha256(file_path):
    """Calculates SHA-256 hash to verify backup integrity."""
    hasher = hashlib.sha256()
    with open(file_path, "rb") as f:
        while chunk := f.read(65536):
            hasher.update(chunk)
    return hasher.hexdigest()

def execute_programmed_backup():
    """Automates directory compression, timestamping, and verification."""
    # 1. Ensure vault directory exists
    os.makedirs(BACKUP_VAULT_DIR, exist_ok=True)

    # 2. Construct timestamped archive filename
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    archive_base_name = os.path.join(BACKUP_VAULT_DIR, f"db_backup_{timestamp}")

    # 3. Create compressed Zip/Tar archive
    print(f"[START] Archiving data from: {SOURCE_DATA_DIR}")
    archive_path = shutil.make_archive(archive_base_name, 'zip', SOURCE_DATA_DIR)

    # 4. Generate integrity hash for disaster recovery validation
    digest = generate_file_sha256(archive_path)

    # 5. Write audit log
    log_file = os.path.join(BACKUP_VAULT_DIR, "backup_audit_log.txt")
    with open(log_file, "a") as log:
        log.write(f"TIMESTAMP: {timestamp} | FILE: {os.path.basename(archive_path)} | SHA256: {digest}\n")

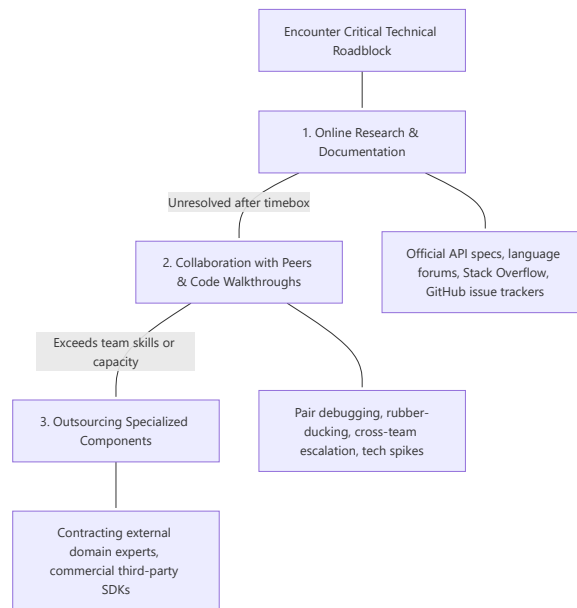
    print(f"[SUCCESS] Programmed backup completed: {archive_path}")
    print(f"[VERIFY] SHA-256 Checksum: {digest}")

if __name__ == "__main__":
    execute_programmed_backup()

```

15.6 Strategies for Overcoming Technical Difficulties

Even the most experienced software engineers encounter technical obstacles—such as undocumented API bugs, unexpected race conditions, or third-party library deprecations. Software engineers follow a structured escalation path to resolve difficulties:



1. Strategic Online Research

- **Navigating Developer Ecosystems:** Engineers must move past generic search engine queries to target authoritative technical resources: official language documentation, vendor RFC specifications, and issue trackers on platforms like GitHub and Stack Overflow.
- **Minimal Reproducible Examples (MRE):** When researching or posting about an issue online, engineers isolate the buggy behavior into the smallest possible standalone code snippet. Stripping away application-specific bloat often reveals the root cause of the bug before external advice is even needed.

2. Collaboration with Peers

- **Pair Debugging & "Rubber Ducking":** Explaining the algorithm's intended logic line by line to a peer (or even talking through the code aloud) forces the engineer to articulate assumptions, often uncovering edge-case logic oversights.
- **Technical Spikes:** When facing deep architectural uncertainty, teams assign one developer to write a throwaway prototype called a **spike**. The spike explores the integration or library in isolation, verifying feasibility before touching production code.

3. Outsourcing Solutions

- **Commercial Third-Party Services:** Building every subsystem from scratch is inefficient and costly. When a project requires specialized capabilities outside the team's core competencies (such as complex credit card handling, optical character recognition, or global SMS routing), teams often **outsource** those functions to validated third-party platforms (e.g., Stripe for payments, Twilio for telecommunications).

- **External Specialists:** If a project encounters a rare, high-consequence requirement (such as formal penetration testing, cryptographic key auditing, or low-level assembly driver optimization), organizations contract external domain specialists to audit or build the component.

Chapter 15 Practice Questions

Multiple Choice Questions

Question 15.1 [Source: 2025 HSC Exam, Q4]

Examine the following two Python fragments designed to sum an array of numbers:

Fragment 1:

```
numbers = [1, 2, 3, 4, 5]
total = 0
for num in numbers:
    total += num
print(total)
```

Fragment 2:

```
numbers = [1, 2, 3, 4, 5]
print(sum(numbers))
```

Fragment 2 is an example of:

- (A) recursive code.
- (B) list comprehensions.
- (C) language-dependent code optimisation.
- (D) using a hash table to create shorter solutions.

Question 15.2 [Source: 2025 HSC Exam, Q14]

Consider the following algorithm fragment:

```
FOR index = 5 TO 3 STEP -1
    word[index] = word[index - 2]
NEXT index
```

The array `word` initially contains: `['S', 'M', 'A', 'R', 'T']` (indexes 1 to 5). What are the contents of `word` after the code executes?

- (A) `['S', 'M', 'A', 'R', 'T']`
- (B) `['S', 'M', 'S', 'M', 'A']`
- (C) `['S', 'M', 'A', 'M', 'S']`
- (D) `['A', 'R', 'T', 'R', 'T']`

Short Answer Questions

Question 15.3 (3 marks) [Source: 2025 HSC Exam, Q17]

An algorithm is required to calculate an excursion hazard score from an array of 10 risk questions (each scored 1 to 5) stored in an array `Risks`. The total score maps to:

- Low: Total less than 20
- Medium: Total between 20 and 35
- High: Total above 35

Using pseudocode with a **repeat-until** loop, write an algorithm that calculates the total risk score, determines the risk rating, and displays the result.

Question 15.4 (5 marks) [Source: 2025 HSC Exam, Q26]

As part of a game, a Python program is needed to place 15 diamonds randomly on a 5×5 grid. The program must generate 15 unique positions and print the 5×5 grid, displaying `'X'` for each diamond and a blank space for empty cells. Write a Python program that completes this task.

Chapter 15 Marking Guidelines & Exemplars

Question 15.1 Solution

- **Correct Answer:** (C) language-dependent code optimisation.
- *Rationale:* Replacing an explicit for-loop with Python's built-in `sum()` function uses the runtime's compiled C implementation to execute the addition faster, demonstrating language-dependent code optimization.

Question 15.2 Solution

- **Correct Answer:** (B) `['S', 'M', 'S', 'M', 'A']`.
- *Rationale:* Tracing backwards:
 - `index = 5 : word[5] = word[3]` ('A') → Array: S, M, A, R, A
 - `index = 4 : word[4] = word[2]` ('M') → Array: S, M, A, M, A
 - `index = 3 : word[3] = word[1]` ('S') → Array: S, M, S, M, A

Question 15.3 Marking Guide

MARKING CRITERIA	MARKS
• Provides correct pseudocode using a <code>REPEAT ... UNTIL</code> loop iterating through all 10 array elements. • Calculates the cumulative sum and evaluates boundary logic (<code>< 20</code>, <code>20..35</code>, <code>> 35</code>) with correct display output.	3

MARKING CRITERIA	MARKS
Provides pseudocode that calculates the sum and evaluates selection logic, with minor syntax errors in loop termination.	2
Shows basic understanding of repetition and selection structures.	1

Band 6 Sample Exemplar:

```

BEGIN ExcursionRiskRating
    total_score = 0
    index = 0

    REPEAT
        total_score = total_score + Risks[index]
        index = index + 1
    UNTIL index = 10

    IF total_score > 35 THEN
        rating = "High"
    ELSEIF total_score >= 20 THEN
        rating = "Medium"
    ELSE
        rating = "Low"
    ENDIF

    DISPLAY "Total Excursion Risk Score: ", total_score
    DISPLAY "Assigned Risk Rating: ", rating
END ExcursionRiskRating

```

Question 15.4 Marking Guide

MARKING CRITERIA	MARKS
- Writes a complete, correct Python program that: - Generates 15 unique random positions (1–25) without duplicates. - Formats output into a 5×5 grid with cell dividers. - Uses appropriate data structures and looping constructs.	5
Generates random positions and prints a grid with minor formatting or duplicate handling errors.	4
Populates an array and prints a grid with logic errors.	3
Shows partial logic for random generation or grid iteration.	2
Shows basic understanding of loops and random libraries.	1

Band 6 Sample Exemplar:

```
import random

# Generate 15 unique random cell positions between 1 and 25
diamond_positions = set()
while len(diamond_positions) < 15:
    diamond_positions.add(random.randint(1, 25))

# Render the 5x5 grid
cell_num = 1
for row in range(5):
    row_display = "|"
    for col in range(5):
        if cell_num in diamond_positions:
            row_display += " X |"
        else:
            row_display += "   |"
        cell_num += 1
    print(row_display)
```

Chapter 16 - Quality Assurance, Verification & Project Evaluation

Syllabus Dot Points Covered:

- *Research and planning* — Investigate how software engineering solutions are quality assured: defining criteria on which quality will be judged, ensuring requirements are met using a continual checking process, addressing compliance and legislative requirements
- *Testing and evaluating* — Apply methodologies to test and evaluate code
- *Course Specifications* — Methods for testing a system: functional testing, acceptance testing, live data, simulated data, beta testing, volume testing
- *Testing and evaluating* — Evaluate the effectiveness of a software engineering solution: developing a report to synthesise feedback, developing a test plan, testing data used/generated based on path and boundary testing, comparing actual output with expected output
- *Research and planning* — Explore communication issues associated with project work: involving and empowering the client, enabling feedback, negotiating

16.1 Quality Assurance (QA) Frameworks

Quality in software engineering is not accidental; it is engineered through structured frameworks.

Syllabus Definition

Quality Assurance (QA): The planned, systematic pattern of all actions necessary to provide adequate confidence that a software product conforms to established technical requirements, performance specifications, and legislative standards.

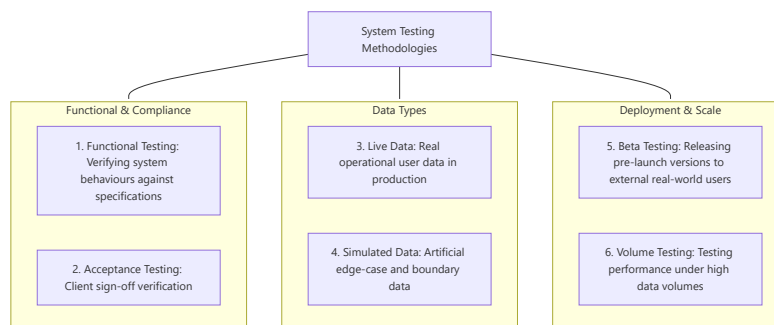
Dimensions of Software Quality Assurance

1. **Defining Quality Criteria:** Software quality is evaluated across explicit metrics:
 - *Correctness & Functionality:* Does the system produce valid outputs for all allowable inputs?
 - *Reliability & Fault Tolerance:* Can the software recover gracefully from unexpected hardware disconnects or invalid data entries?
 - *Maintainability:* Is code modular, well-commented, and aligned with styling standards so future developers can update it easily?
 - *Usability:* Does the interface conform to W3C accessibility guidelines with low cognitive friction?

2. **Continual Checking Processes:** Integrating code reviews, static automated analysis, and continuous unit testing into sprint cycles rather than waiting for post-development testing.
 3. **Compliance and Legislative Requirements:** Ensuring the system complies with laws, such as the *Copyright Act 1968* (respecting third-party code licenses), the *Privacy Act 1988* (protecting user data), and relevant workplace health and safety (WHS) laws.
-

16.2 System Testing Methodologies

The **NESA Course Specifications** identify six core testing methodologies:



1. **Functional Testing:** Validates that each specific software capability operates in strict accordance with the functional requirements specification. Tests inputs and checks that corresponding outputs match expected results (e.g., verifying that entering valid credit card details produces an authorization code).
 2. **Acceptance Testing (User Acceptance Testing - UAT):** Conducted by the end user or client in a staging environment to determine whether the software fulfills business requirements and contractual criteria before final payment and deployment.
 3. **Live Data:** Real, operational data generated by real users in production. Useful for confirming that system integrations handle unexpected real-world user behaviors, though sensitive PII must be protected.
 4. **Simulated Data:** Artificially generated data designed to thoroughly exercise all execution paths, boundary values, and abnormal conditions before live deployment (e.g., generating 50,000 artificial sensor readings to test crash tolerance).
 5. **Beta Testing:** Releasing a nearly finished, pre-release version of software to an external audience of real users outside the development team. Beta testers use the software in varied physical hardware and operating environments, identifying edge-case bugs that internal QA teams missed.
 6. **Volume (Load/Stress) Testing:** Evaluates software performance under extreme conditions by subjecting the application, database, and network to heavy loads (e.g., streaming thousands of simultaneous video requests to ensure the database does not exhaust connection pools).
-

16.3 Test Plan Design: Boundaries, Paths, and Data Types

A **Test Plan** is a formal document detailing the test data, execution paths, expected outputs, and actual outcomes used to verify software correctness.

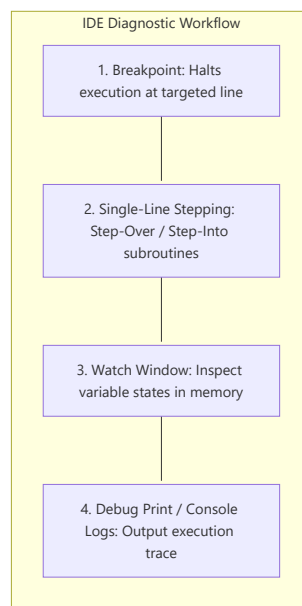
Categories of Test Data

- **Normal (Typical) Data:** Valid, expected values falling well within allowable ranges (e.g., entering an age of 28 for an age field).
- **Boundary (Edge) Data:** Values at the minimum and maximum boundaries of acceptable ranges, where off-by-one errors frequently occur (e.g., testing 17, 18, 64, and 65 for a system that classifies ages 18..64 as working age).
- **Abnormal (Faulty/Invalid) Data:** Data outside allowable types or physical ranges, designed to test input validation and error handling (e.g., submitting "twenty" or -5 into an age field).

TEST ID	TEST DATA INPUT	REASON FOR INCLUSION	EXPECTED OUTPUT	ACTUAL OUTPUT	PASS / FAIL STATUS
T-01	Mark = 85	Typical value in High range (>80)	Grade = "H"	Grade = "H"	PASS
T-02	Mark = 80	Boundary check (exact threshold)	Grade = "H"	Grade = "H"	PASS
T-03	Mark = 79	Boundary check (just below 80)	Grade = "P"	Grade = "P"	PASS
T-04	Mark = 50	Boundary check (exact pass limit)	Grade = "P"	Grade = "P"	PASS
T-05	Mark = 49	Boundary check (just below pass)	Grade = "N"	Grade = "N"	PASS
T-06	Mark = -5	Abnormal negative data	Error: Invalid Mark	Error: Invalid Mark	PASS
T-07	Mark = "eighty"	Abnormal non-numeric data	Error: Type Mismatch	Error: Type Mismatch	PASS

16.4 Debugging Tools in Integrated Development Environments (IDEs)

When testing uncovers faults, engineers use IDE debugging tools to isolate logic and runtime errors:



- **Breakpoints:** Markers set on specific source code lines that tell the IDE to pause program execution immediately before that line runs, freezing application memory for inspection.
- **Single-Line Stepping (Step Into / Step Over):** Allows the programmer to execute code line by line, watching variable transformations and conditional branches unfold in real time.
- **Watch Windows:** An IDE window that displays the contents of specific variables or expressions in memory, updating automatically as the code executes.
- **Debugging Output Statements:** Temporary logging commands (e.g., `print(f"[DEBUG] count={count}"))` placed at key checkpoints to verify whether specific branches were reached during execution.

16.5 Client Communication and Post-Implementation Evaluation

The success of a software engineering project depends on clear communication between technical teams and non-technical clients:

- **Involving and Empowering the Client:** Engaging clients in weekly sprint demonstrations and design walkthroughs ensures the product aligns with business needs and gives clients ownership of product decisions.
- **Enabling Feedback:** Providing accessible feedback mechanisms (user bug trackers, feedback surveys, staging environments) allows clients to communicate usability concerns early.
- **Negotiation:** When technical constraints, schedule changes, or budget limits conflict with client requests, engineers must negotiate changes to scope or delivery timelines constructively.
- **Post-Implementation Evaluation:** At project completion, the team produces an evaluation report that synthesizes user feedback, compares actual system performance against initial specifications, and identifies lessons learned to improve future development cycles.

Chapter 16 Practice Questions

Multiple Choice Questions

Question 16.1 [Source: APHS 2025 Trial, Q1 / Q3]

Which pair of testing techniques should be used during test planning to thoroughly evaluate an algorithm's decision boundaries and execution paths?

- (A) Data validation and Data verification
- (B) Path testing and Boundary testing
- (C) Beta testing and Volume testing
- (D) Direct cutover and Phased testing

Question 16.2 [Source: 2025 HSC Exam, Q15b]

A developer encounters persistent logic errors in a complex tax calculation program. Which two debugging tools within an IDE would be most effective for inspecting how variable states change during execution?

- (A) CSS resets and Web App Manifests
- (B) Breakpoints and Variable Watches
- (C) Gantt charts and Storyboards
- (D) Decision trees and DFDs

Short Answer Questions

Question 16.3 (3 marks) [Source: 2025 NESA Sample Exam, Q11]

Consider the following algorithm:

```
BEGIN responsiveScreen
  Get ScreenSize
  IF ScreenSize >= 1024 THEN
    Viewport = 'Desktop'
  ELSEIF ScreenSize >= 768 THEN
    Viewport = 'Tablet'
  ELSE
    Viewport = 'Phone'
  ENDIF
  Display "Display set to ", Viewport
END responsiveScreen
```

Construct a test data table that will thoroughly test this algorithm based on boundary and path coverage. Include expected outputs and reasons for inclusion.

Question 16.4 (3 marks) [Source: HSC Style Questions 2025, Q10c / Sample Trial 2025, Q22a]

A software startup hires an offshore development team to code its customer-facing web platform. Explain THREE communication challenges that could arise from managing stakeholders across geographically dispersed teams, and describe how these challenges can impact project delivery.

Chapter 16 Marking Guidelines & Exemplars

Question 16.1 Solution

- **Correct Answer:** (B) Path testing and Boundary testing.
- **Rationale:** Path testing ensures every programmatic branch executes at least once, while boundary testing targets edge values where logical comparison errors most frequently occur.

Question 16.2 Solution

- **Correct Answer:** (B) Breakpoints and Variable Watches.
- **Rationale:** Breakpoints halt execution at specific lines, and variable watches let developers observe live variable values in memory as the code steps forward.

Question 16.3 Marking Guide

MARKING CRITERIA	MARKS
• Constructs a complete test data table testing all paths (Desktop, Tablet, Phone) and all boundary values (1024, 768). • Correctly specifies test inputs, expected outputs, and valid justifications.	3
• Provides test data that covers most paths and boundaries with expected outputs.	2
• Provides basic test inputs with partial outputs.	1

Band 6 Sample Exemplar:

TEST INPUT (SCREENSIZE)	EXPECTED OUTPUT (VIEWPORT)	REASON FOR INCLUSION / TESTING JUSTIFICATION
1024	'Desktop'	Boundary Value: Exact minimum threshold for Desktop branch.
1023	'Tablet'	Boundary Value: Just below Desktop threshold; verifies upper edge of Tablet branch.
768	'Tablet'	Boundary Value: Exact minimum threshold for Tablet branch.
767	'Phone'	Boundary Value: Just below Tablet threshold; verifies upper edge of Phone branch.

TEST INPUT (SCREENSIZE)	EXPECTED OUTPUT (VIEWPORT)	REASON FOR INCLUSION / TESTING JUSTIFICATION
400	'Phone'	Normal Value: Typical value testing lower Phone path.

Question 16.4 Marking Guide

MARKING CRITERIA	MARKS
- Explains THREE distinct communication challenges resulting from geographically dispersed teams (e.g. time zone differences, feedback delays, cultural/language nuances in requirements). - Clearly links these communication issues to project delivery impacts (delays, misunderstandings, rework).	3
Explains two communication issues with sound detail.	2
Outlines an isolated communication challenge.	1

Band 6 Sample Exemplar:

- Time Zone Discrepancies:** Large time zone differences reduce the window for real-time collaboration. Clarifications on critical specifications or bug reports can face 12-to-24-hour delays, slowing down sprint iterations.
- Feedback Delays and Asynchronous Misunderstandings:** Relying on written tickets or asynchronous chats rather than face-to-face walkthroughs increases the risk of misinterpreted requirements. Developers may spend days building features based on flawed assumptions, requiring costly rework.
- Negotiation and Cultural Nuance:** Negotiating scope changes or resolving technical disagreements remotely can lead to frustration and misaligned expectations if teams do not establish shared communication protocols and structured daily stand-ups.