

Essentials: know these by heart

Software Engineering HSC · Tue 27 Oct 2026, 9.50 am · top picks only (★ on the web page) · full list with Test me mode: tk-study.pages.dev/software-engineering/essentials/

1. Exam format and command words

1. **When and how long.** Tue 27 Oct 2026, 9.50 am to 12.20 pm. 10 min reading + 2 h 20 min writing = 2 h 30 min, 80 marks (about 1.75 min per mark). Sat online with wired headphones.
2. **Paper shape (2025).** Q1–14 objective (16 marks), Q15–27 written, coded, drawn or extended (64 marks).
3. **Identify.** Recognise and name. A word or phrase is enough.
4. **Outline.** Sketch in general terms; a short statement per main point.
5. **Describe.** Give characteristics and features: what it is and what it does, with a detail.
6. **Compare.** Similarities AND differences (both, whereas, in contrast).
7. **Explain.** Cause and effect: the how or why (because, therefore, which means).
8. **Discuss.** Give points for and against: a positive AND a negative.
9. **One point per mark.** About three developed points for 3 marks. Do not write an essay for 3 marks.
10. **Use the scenario.** Name a detail from the stimulus. Marking ladders give the top mark only when the answer links to the scenario.
11. **Code answers (marker feedback).** Call your function after defining it; indent correctly; put quotation marks around text in SQL.

2. Year 11: fundamentals, OOP, mechatronics

1. **Eight development steps.** Requirements definition, determining specifications, design, development, integration, testing and debugging, installation, maintenance.
2. **Three control structures.** Sequence, selection, iteration (repetition). Every algorithm is a combination of these.
3. **Four paradigms.** Imperative (step-by-step commands that change state), object-oriented (objects from classes), logic (facts and rules), functional (functions and recursion, avoid changing state).
4. **Binary and hex.** Binary place values 128 64 32 16 8 4 2 1 (1011 = 11). One hex digit = 4 bits; hex digits 0–9, A–F (0xB3 = 1011 0011 = 179).
5. **Two's complement.** Leftmost bit is the sign (1 = negative). To negate: invert all bits then add 1 (5 = 00000101 → 11111011 = -5).
6. **Class and object.** A class is the blueprint (attributes and methods). An object is an instance of a class with its own attribute values.
7. **Encapsulation.** Bundle data and the methods that use it in a class and restrict direct access (private attributes, getters and setters).
8. **Abstraction.** Hide complex detail and expose only what is needed (call `account.deposit()` without knowing how it works).
9. **Inheritance.** A subclass takes the attributes and methods of a superclass and can add or change them (code reuse; an "is-a" relationship).
10. **Polymorphism.** Different classes give their own version of the same method name, so one call behaves differently for different objects (`speak()` for Dog and Cat).
11. **Three error types.** Syntax (breaks language rules, will not run), logic (runs, wrong result), runtime (fails while running, e.g. divide by zero).
12. **Sensor, actuator, end effector.** Sensor: input that detects the environment. Actuator: output that causes movement (motor, servo). End effector: device at the end of a robot arm (gripper).
13. **Open v closed loop.** Open loop: output set without checking the result (timer sprinkler). Closed loop: sensor feedback adjusts the output to reach a target (thermostat, cruise control).

3. Secure software architecture

1. **Confidentiality.** Only authorised people can read the information (encryption, access controls).
2. **Integrity.** Data is accurate and unchanged by unauthorised parties (hashing, checksums, digital signatures, validation).
3. **Availability.** Authorised users can reach the system and data when needed (backups, redundancy, protection from denial of service).
4. **Authentication.** Verifying a user or system is who they claim to be (password, MFA, certificate). Answer used in 2025 Q6.
5. **Authorisation.** Deciding what an authenticated user may do (roles, least privilege). Authentication comes first.
6. **Security by design.** Security is built in from the start of development, not added at the end.
7. **Cryptography.** Converts plain text to cipher text with an algorithm and a key so only key holders can read it.
8. **Symmetric v asymmetric.** Symmetric: one shared key both ways, fast. Asymmetric: public key encrypts, private key decrypts (or signs); solves key sharing but slower.
9. **Hashing.** One-way function giving a fixed-length hash; same input = same hash; cannot be reversed. Stores passwords (with a salt) and checks integrity.
10. **Sandboxing.** Running code in an isolated, restricted environment so it cannot harm the rest of the system (browser tabs, containers).
11. **Privacy by design: three parts.** Proactive not reactive; embed privacy into design (collect only needed data, private defaults, encrypt, retention limits); respect for user privacy (consent, control).
12. **Regulatory compliance.** Meeting laws, regulations and industry standards. It is about protecting sensitive data and meeting legal standards, NOT limiting updates, bypassing security or cutting testing.
13. **SAST.** Static: analyses source code WITHOUT running it, early, white box; can give false positives.
14. **DAST.** Dynamic: tests the RUNNING app from outside, later, black box (no source code). Finds injection, XSS, misconfiguration.
15. **Vulnerability assessment v penetration test.** Assessment: scan, list and rank weaknesses (does not exploit). Pen test: authorised simulated attack that exploits them.
16. **Business continuity v disaster recovery.** Continuity keeps the business operating; disaster recovery restores IT systems and data. RTO = how long until back; RPO = how much data loss (in time) is acceptable.
17. **Defensive input handling.** Never trust input: validate (type, range, length, format), sanitise (neutralise dangerous characters), error-handle (try/except, safe message, log detail privately).
18. **API risk.** Excessive data exposure = exposing more data than intended. Fixes: return only needed fields, authenticate and authorise every request, validate input, HTTPS, rate limiting, generic error messages.
19. **Session management.** Track state with a random, unpredictable session ID in a cookie: expiry/timeout, HTTPS only (Secure, HttpOnly), destroy on logout. Limit session data to keep it efficient.
20. **Broken authentication.** Weak login or session handling lets attackers take over accounts. Fix: strong passwords and hashing, MFA, logout or rate limiting, session timeout.
21. **XSS.** Attacker's script runs in other users' browsers via a page. Fix: sanitise input, escape/encode output, Content Security Policy.
22. **CSRF.** A malicious site makes a logged-in user's browser send an unwanted request. Fix: anti-CSRF tokens, check the origin, SameSite cookies.

23. **Race condition.** Two processes use shared data at the same time and the result depends on timing (two purchases of the last item). Fix: locks, atomic operations, transactions.

4. Programming for the web

1. **Packet.** A header (source and destination IP, sequence number, protocol) plus a payload (the data); reassembled in order at the destination using the sequence numbers.
2. **IP address and DNS.** IPv4 = four numbers 0–255 (32 bits) identifying a device. DNS translates a domain name to its IP address (port 53).
3. **Ports (web).** HTTP 80 (not encrypted); HTTPS 443 (HTTP over SSL/TLS).
4. **Ports (files and email).** FTP 21; SMTP 25 sends mail; POP3 110 downloads mail to one device; IMAP 143 keeps mail on the server, synced.
5. **Request flow.** Browser asks DNS for the IP, opens a TCP connection (80 or 443), sends the HTTP(S) request, the server returns a response (status code and content), the browser renders it.
6. **SSL/TLS certificate.** Issued by a trusted certificate authority; proves the site's identity and holds its public key; enables HTTPS (authentication of the server).
7. **Hash v digital signature.** Hash value: changes if the data changes (integrity). Digital signature: hash encrypted with the sender's private key; checked with the public key (origin and integrity).
8. **Front end and back end.** Front end (client): HTML structure, CSS style, JavaScript behaviour. Back end (server): application logic, authentication, database access (Python Flask/Django, Node.js, PHP).
9. **SQL v NoSQL.** SQL: relational tables, fixed schema, good for transactions. NoSQL: flexible schema, scales horizontally, suits big and unstructured data.
10. **ORM v SQL.** SQL = control and performance. ORM = convenience and portability: you use objects and the ORM writes the SQL (2025 Q9).
11. **Data mining and metadata.** Data mining finds non-obvious patterns in large data. Metadata describes other data (title, duration, time, location). Metadata lets mining find patterns and predict demand (2025 Q18).
12. **W3C and WAI.** W3C: international web standards body (HTML, CSS). WAI: its accessibility initiative, producing WCAG, ATAG and WAI-ARIA.
13. **WCAG colour contrast.** 4.5:1 for normal text, 3:1 for large text. Do not rely on colour alone.
14. **PWA.** Web app that installs, works offline and loads fast. Built with a service worker (background script that serves from cache or network), a web app manifest and HTTPS.
15. **CSS.** Controls presentation separately from HTML. One external style sheet = consistent look; media queries adapt to different devices. selector { property: value; }
16. **Load-time methods.** Minification, image optimisation, lazy loading, caching, CDN, async/defer scripts, back-end tuning (select only needed columns, index tables).

5. Software automation and machine learning

1. **AI v ML.** ML is a subset of AI: algorithms that learn patterns from data. All ML is AI; not all AI is ML.
2. **Supervised.** Trained on LABELLED data to predict labels (classification, regression).
3. **Unsupervised.** Finds patterns in UNLABELLED data (clustering, anomaly detection).
4. **Reinforcement.** An agent learns by trial and error from rewards and penalties.
5. **Memory hook.** Labels = supervised. No labels = unsupervised. Few labels = semi. Rewards = reinforcement.
6. **Linear regression.** Predicts a continuous number by fitting a straight line ($y = mx + c$).
7. **Logistic regression.** Despite the name it CLASSIFIES: predicts the probability (0 to 1, sigmoid) of a category, usually yes/no (2025 Q12).
8. **Choosing a model.** A number = linear or polynomial. A category or yes/no = logistic or KNN. Straight trend = linear; curved = polynomial.
9. **Neural network layers.** Input layer, hidden layers (more = deep), output layer. Connections have weights; neurons use an activation function.
10. **Training cycle.** Forward propagation → loss (prediction v correct answer) → backpropagation (send error back, update weights) → repeat over many epochs.
11. **MLOps three stages.** Design (define the problem and success metrics); Model development (data wrangling, feature engineering, training, testing); Operations (deployment, support, monitoring).
12. **DevOps.** Combines development and operations for faster, more reliable delivery. Principles: rapid feedback, automation of repetitive tasks, collaboration (2025 Q11).
13. **RPA v BPA.** RPA: software bots mimic a user on individual repetitive tasks. BPA: automates a whole end-to-end process.
14. **Human bias.** Explicit (intentional) or implicit (unconscious) bias can enter at design, data collection and labelling.
15. **Dataset bias.** Training data that is not diverse or representative, so the model performs poorly for some groups.
16. **Bias fixes.** Diverse representative data, dataset audits, diverse teams, clear labelling guidelines with several reviewers, testing across groups.
17. **Impact discussions (5 marks).** Give a positive AND a negative, each tied to the scenario. Areas: worker safety, people with disability, employment, efficiency and environment, economy.

6. Software engineering project

1. **Functional v performance requirements.**
Functional: what the system must DO.
Performance: how WELL (speed, capacity, reliability).
2. **Direct implementation.** Old system off, new on at once. Cheapest and quickest; highest risk (no fallback).
3. **Phased implementation.** New system introduced in stages (module by module). Lower risk, takes longer.
4. **Parallel implementation.** Old and new run side by side and outputs are compared. Safest (backup) but costs more. Critical systems.
5. **Pilot implementation.** Installed for a small group or one location first, then rolled out to everyone.
6. **Choosing an implementation method.** Critical = parallel. Small, urgent or low risk = direct. Large with separable parts = phased. Many sites = pilot.
7. **Waterfall.** Sequential, plan-driven: each stage finishes before the next. Good for fixed requirements; inflexible, client sees product late.
8. **Agile.** Iterative: working software in short sprints with constant client feedback. Flexible; less predictable cost and less documentation.
9. **WAgile.** Hybrid: Waterfall-style planning and design up front, Agile sprints for development.
10. **Gantt chart.** Bar chart of tasks against time showing start/end, durations and dependencies.
11. **Data dictionary columns.** Variable, data type, format for display, size in bytes, size for display, description, example, validation.
12. **Choosing data types.** Name = string; whole count = integer; price or mark = real; yes/no = Boolean; birth date = date and time.
13. **Testing levels.** Unit (one part), integration/subsystem (modules together), system (whole system v requirements), acceptance (client confirms), beta (real users trial a near-final version).
14. **Test data.** Boundary values (range 1–10: test 0, 1, 10, 11), path coverage (every branch/loop runs), faulty/abnormal data (wrong type, empty, negative).
15. **Debugging tools.** Breakpoint (pause on a line), single-line stepping, watches (track variables), debugging output statements (print), IDE debugger.
16. **Maintenance types.** Corrective (fix bugs), adaptive (new environment, law or hardware), perfective (improve performance or add features).
17. **Stub v driver.** Stub: stand-in for a low-level module not built yet (returns a fixed value). Driver: temporary code that calls a module that has no mainline yet.

7. Diagram notation (symbol meanings)

1. **Flowchart symbols.** Terminator (rounded) = Start/End; rectangle = process; parallelogram = input/output; diamond = decision with labelled Yes/No arrows. Read top to bottom, left to right.
2. **DFD symbols.** Circle = process (verb phrase, at least one input and one output); open rectangle = data store; rectangle = external entity; labelled arrow = data flow (name the data, not the action).
3. **DFD rule.** Data moves between a store or entity only via a process, never directly (no entity-to-store flow).
4. **Level 0 v Level 1.** Level 0 (context): one process for the whole system, entities and labelled flows only, no stores. Level 1: sub-processes and data stores; inputs and outputs must balance with Level 0.
5. **Structure chart symbols.** Top box = mainline; sub-modules below; EMPTY circle = data passed; FILLED circle = flag (control value); diamond = decision; curved arrow = repetition.
6. **Class diagram.** Three-part box: name, attributes, methods. Inheritance arrow from child to parent (shared attributes only in the parent); show multiplicity (1, 0..*).

8. Python, SQL and pseudocode must-know

1. **input() is a string.** Convert numbers with int() or float().

```
age = int(input("Enter age: "))  
print(f"Hello {name}")
```
2. **for and range.** range(1, 6) gives 1,2,3,4,5 (stop is excluded). Check off-by-one.

```
for i in range(1, 6):  
    print(i)
```
3. **Selection.** if / elif / else, each with a colon and an indented body.
4. **Strings and slicing.** s[:3] = first 3 characters, s[-1] = last, len(s), "war" in s.

```
expected = uName[:3] + "123" # 2025 Q19
```
5. **Function: define, return, CALL.** Markers noted students defined functions but never called them. Call it and print or use the result.
6. **random.randint.** Inclusive of both ends.

```
import random  
n = random.randint(1, 25)
```
7. **try / except.** Catch a runtime error and give a safe message.
8. **File handling.** "w" overwrites, "a" appends; with closes the file automatically.
9. **Class, constructor, method.** __init__ sets attributes with self; methods take self.
10. **Inheritance in Python.** Subclass in brackets; super().__init__() calls the parent constructor; redefine a method to override.
11. **Parameterised query.** The ? placeholder sends user input as DATA, so it can never run as SQL (stops SQL injection). Never join input into the SQL string.

```
cur.execute("SELECT * FROM Student  
WHERE Name = ?", (user_input,))  
conn.commit() # needed after  
INSERT/UPDATE/DELETE
```
12. **SQL clause order.** SELECT, FROM, WHERE, GROUP BY, ORDER BY. Quote text and date values. WHERE filters rows before grouping.
13. **SQL aggregates and AS.** SUM, AVG, COUNT, MAX, MIN; AS renames a field for display.
14. **SQL join.** Match the primary key of one table to the foreign key in the other; without the matching condition you get every combination of rows.
15. **SQL insert, update, delete.** UPDATE and DELETE need WHERE, or they change every row.
16. **Primary key and foreign key.** Primary key uniquely identifies each row. Foreign key holds another table's primary key. One-to-many: foreign key on the many side; many-to-many needs a linking table.
17. **Pseudocode rules.** Keywords in CAPITALS; every BEGIN has an END, IF has ENDIF, WHILE has ENDWHILE, FOR has NEXT, CASEWHERE has END CASE; indent every body.
18. **Pseudocode loops.** WHILE checks first (body may run zero times). REPEAT...UNTIL runs at least once and stops when the condition is true. FOR counts.