

YEAR 12: COURSEWORK / NEA

Main focus: OCR Programming Project / NEA. The NEA stages remain the organising spine; the Component 01 and 02 theory needed to complete each stage is explicitly aligned below. Remaining examined theory is mapped and assessed separately.

	Autumn 1	Autumn 2	Spring 1	Spring 2	Summer 1	Summer 2
	UNIT OF WORK: NEA Introduction, project suitability and initial Analysis	UNIT OF WORK: NEA Analysis: Investigation, requirements and success criteria	UNIT OF WORK: NEA Design: decomposition, solution structure and initial interface planning	UNIT OF WORK: NEA Design: Interface design, usability and algorithm design	UNIT OF WORK: NEA Design: technical choices, methodology and testing plans	UNIT OF WORK: NEA Development: iterative implementation and development evidence
KNOWLEDGE	Main NEA focus - Introduce the OCR Programming Project requirements and the full project journey: Analysis, Design, Development, Testing and Evaluation. - Select or refine a project that solves a genuine problem, has an identifiable end user or stakeholder, is sufficiently complex and is realistic to complete. - Define the problem, current system or current approach, key weaknesses, project aims, objectives and initial solution ideas. - Explain why the problem is suitable for a computational solution and consider project feasibility, constraints and risks. Intended outcome An approved project proposal and a secure first draft of the problem-definition section.	Main NEA focus - Identify stakeholders and gather primary evidence through interviews, questionnaires, observations or structured discussions. - Research the current system and comparable products, identifying strengths, weaknesses, inefficiencies, useful features and limitations. - Analyse findings rather than simply presenting evidence, and explain how the research affects the proposed solution. - Produce functional and non-functional requirements, essential features, realistic limitations, hardware/software requirements and measurable success criteria. - Compile the Analysis into a coherent formal report with clear links between evidence, requirements and proposed features. Intended outcome A complete Analysis section supported by evidence and measurable requirements.	Main NEA focus - Review and improve the completed Analysis before beginning detailed Design. - Decompose the solution into modules, procedures, functions, classes or subsystems and justify why each component is required. - Identify inputs, processes, outputs, dependencies and the way components will communicate or integrate. - Explain and justify the overall structure of the proposed solution. - Begin annotated interface designs that reflect the target user, requirements and planned functionality. Intended outcome A justified solution architecture and initial interface designs.	Main NEA focus - Complete annotated interface designs covering navigation, consistency, readability, accessibility, feedback, input methods, validation and error prevention. - Design an algorithm for each significant element of functionality, normally using structured pseudocode. - For each algorithm, describe its purpose, explain how it works, justify key decisions and link it to the relevant requirement or success criterion. - Check that the complete solution has been covered and that large processes have been divided into manageable algorithms. Intended outcome The principal interfaces and algorithms required to implement the solution.	Main NEA focus - Identify and justify key variables, constants, data types, scope, classes and data structures. - Select appropriate structures such as lists, arrays, dictionaries, records, files or databases according to project need. - Explain and justify the proposed development methodology, including iterative, incremental, modular and prototyping approaches. - Plan development-stage testing using normal, boundary, erroneous/invalid and extreme data where relevant. - Create acceptance tests linked directly to requirements and success criteria. - Write a comprehensive Design conclusion showing that the solution is coherent, complete, testable and realistic to implement. Intended outcome A completed, implementable Design with full traceability from requirements to planned testing.	Main NEA focus - Begin implementing the solution in manageable increments and produce working prototypes or components. - Document meaningful development stages rather than presenting pages of unexplained code or retrospective diary entries. - For each stage, record the aim, requirement addressed, planned approach, implementation evidence and explanation of important code. - Test each increment, record expected and actual outcomes, identify errors or weaknesses, take remedial action and retest. - Use the development cycle: Plan -> Implement -> Test -> Identify problem -> Fix -> Retest -> Reflect. - Continue implementation and evidence gathering over the summer and into Year 13. Intended outcome Working components and the first clear Development-story entries showing genuine iteration.
NEA-ALIGNED THEORY	Supporting specification knowledge (Components 01 and 02) - 2.2.2(a-e) Computational methods: features of computationally solvable problems, problem recognition, decomposition, divide and conquer and abstraction. - 2.1.1-2.1.4 Computational thinking: abstract models; inputs, outputs and preconditions; procedural decomposition; decision points, conditions and program flow. - 1.2.2(a) Applications generation: the nature of applications and justification of a suitable application for a specific purpose. Application to the NEA Use this theory to justify computational suitability, identify inputs/processes/outputs, define a realistic scope and show that the proposed project is sufficiently substantial.	Supporting specification knowledge (Components 01 and 02) - 2.1.2(a-b) Thinking ahead: identifying inputs, outputs and preconditions for the proposed solution. - 1.1.3(a-d) Input, output and storage: selecting appropriate devices, memory and storage for a particular solution. - 1.2.2(a-c) Applications generation: application purpose, utilities and open-source versus closed-source choices. - 1.3.2 and 1.3.3(c) Data management, databases and security where relevant to the proposed project. - 1.5.1-1.5.2 Legal, moral, cultural and ethical issues: privacy, copyright, computer misuse, accessibility and responsible handling of user data. Application to the NEA Apply the theory when justifying hardware/software, input/output, data-storage, security, accessibility and legal requirements, as well as realistic limitations and measurable success criteria.	Supporting specification knowledge (Components 01 and 02) - 2.1.3(a-d) Thinking procedurally: identifying solution components, ordering steps and identifying required sub-procedures. - 2.1.2(d) Thinking ahead: the need for reusable program components. - 2.2.1(c-d) Programming techniques: local/global variables, modularity, functions, procedures and parameter passing by value/reference. - 1.2.4(a-e) Programming paradigms: procedural and object-oriented approaches, including classes, objects, methods, attributes, inheritance, encapsulation and polymorphism where appropriate. Application to the NEA Use these concepts to justify the project architecture, define clear module responsibilities and select an appropriate procedural and/or object-oriented structure.	Supporting specification knowledge (Components 01 and 02) - 2.2.1(a-b, d, f) Programming techniques: sequence, selection, iteration, recursion, subroutines and object-oriented techniques. - 2.1.4(a-c) Thinking logically: decision points, logical conditions and how decisions affect program flow. - 2.3.1(a-f) Algorithms: analysis and design, suitability, time/space efficiency, Big O notation and standard algorithms where directly relevant. - 1.5.2 Moral and ethical issues: layout, colour, character sets, accessibility and appropriate interface choices. Application to the NEA Map each substantial requirement to an interface and/or accurate justified algorithm. Apply Big O and standard search, sort or pathfinding algorithms only when genuinely relevant to the project.	Supporting specification knowledge (Components 01 and 02) - 1.4.1(a) Data types: integer, real/floating point, character, string and Boolean, selected according to purpose. - 1.4.2(a-c) Data structures: arrays, records, lists, tuples and advanced structures; creation, traversal, insertion and removal. - 1.2.4(e) Object-oriented programming: classes, objects, methods, attributes, inheritance, encapsulation and polymorphism where relevant. - 1.2.3(a-c) Software development: waterfall, agile, extreme programming, spiral and rapid application development; merits, drawbacks and algorithm use. - 1.3.2(a-f) Database theory where relevant: relational design, keys, ER models, normalisation, SQL, referential integrity and transaction processing.	Supporting specification knowledge (Components 01 and 02) - 2.2.1(a-f) Programming techniques: constructs, recursion, variable scope, modularity, functions/procedures, use of an IDE for debugging and object-oriented techniques. - 1.2.3(a-c) Software development: agile/iterative development and writing/following algorithms. - 1.4.2(c) Data structures: creating, traversing, adding and removing data from the structures chosen in Design. - Project-dependent links: 1.3.1 hashing/encryption; 1.3.2 databases/SQL; 1.3.3 networking/security; and 1.3.4 web technologies, where required by the solution. Application to the NEA Implement one increment at a time, test it, document failures and remedial action, and apply only the additional theory required by the individual technical solution.

	Autumn 1	Autumn 2	Spring 1	Spring 2	Summer 1	Summer 2
	UNIT OF WORK: NEA introduction, project suitability and initial Analysis	UNIT OF WORK: NEA Analysis: investigation, requirements and success criteria	UNIT OF WORK: NEA Design: decomposition, solution structure and initial interface planning	UNIT OF WORK: NEA Design: interface design, usability and algorithm design	UNIT OF WORK: NEA Design: technical choices, methodology and testing plans	UNIT OF WORK: NEA Development: iterative implementation and development evidence
					- Component 03/04, 3.2.3: iterative and post-development test data, validation and acceptance testing. Application to the NEA Justify technical choices and produce traceable plans linking each requirement to variables/classes/data structures, development tests and post-development acceptance tests.	
KEY SKILLS	- Project scoping and feasibility analysis. - Problem identification and stakeholder awareness. - Applying computational thinking to a real problem. - Formal technical writing and verbal project explanation. - Reading the specification and interpreting project expectations.	- Planning and carrying out primary/secondary research. - Analysing qualitative and quantitative findings. - Comparing existing products and drawing justified conclusions. - Writing precise functional/non-functional requirements. - Creating measurable and testable success criteria.	- Redrafting in response to tracker and teacher feedback. - Decomposing a substantial problem into manageable components. - Modelling inputs, processes, outputs and dependencies. - Justifying technical structure and communicating design decisions. - Producing clear annotated interface sketches.	- Applying user-centred design and recognised usability features. - Writing and tracing structured pseudocode. - Explaining selection, iteration, validation and data handling. - Reviewing algorithm coverage and identifying missing functionality. - Peer discussion and oral justification of design choices.	- Selecting and justifying variables, data types and structures. - Planning development and acceptance testing. - Using tables to show traceability between requirements, design and tests. - Evaluating risks, constraints and implementation feasibility. - Producing a coherent conclusion from detailed technical evidence.	- Translating designs into working code. - Explaining code accurately and selecting purposeful evidence. - Testing, debugging, correcting and retesting. - Recording expected/actual results and reflecting on progress. - Maintaining organised files, versions and development records.
CURRICULUM SEQUENCING AND HOW WE BUILD ON SKILLS AND KNOWLEDGE	Builds on KS4 practical programming and computational-thinking skills. Students move from teacher-defined programming tasks to identifying and scoping a substantial problem of their own. A suitable problem and realistic scope must be established before detailed investigation.	Builds on the approved project and identified stakeholder. Research findings are converted into justified requirements and measurable success criteria. The sequence is: investigation -> analysis of findings -> requirements -> success criteria.	Builds directly on the completed Analysis. The requirements define what must be designed, while decomposition establishes how the complete solution will be organised. Students do not begin detailed implementation before the structure is sufficiently clear.	Builds on decomposition and requirements. Each significant feature is represented through interface and/or algorithm design. The sequence is: requirement -> module -> interface/process -> algorithm.	Completes the Design by specifying the data, methodology and testing needed to implement it. Traceability is made explicit: Analysis requirement -> Design component -> algorithm/data structure -> development test -> acceptance test.	Builds from the completed Design into implementation. Students apply theory as required by their own solution and capture evidence while development is taking place. Development continues into Year 13 before final testing and Evaluation.
WIDER APPLICATION AND FUTURE PATHWAYS	Careers: Systems analyst, business analyst, requirements engineer, software developer. Cross-curricular and real-world links: English: formal problem definition and technical explanation. Business: feasibility, stakeholders and project scope. Citizenship: responsible solutions for real users.	Careers: User researcher, business analyst, product manager, data analyst. Cross-curricular and real-world links: English: research questions and analytical writing. Maths: interpreting quantitative evidence. Sociology/Psychology: user needs and research methods.	Careers: Software architect, systems designer, UX/UI designer, database designer. Cross-curricular and real-world links: Design and Technology: decomposition, prototyping and iterative design. English: justification. Maths: logical structures and relationships.	Careers: UX/UI designer, software developer, algorithm engineer, games developer. Cross-curricular and real-world links: Design and Technology: user-centred design. Maths: logic and algorithm tracing. English/Oracy: precise explanations and design reviews.	Careers: Database developer, software tester, quality-assurance analyst, project manager. Cross-curricular and real-world links: Maths: data ranges and boundary values. Business: project methodology and risk. English: precise test plans and technical conclusions.	Careers: Software engineer, DevOps engineer, test engineer, database/web/games developer. Cross-curricular and real-world links: Real-world software development: versioning, debugging, iterative improvement, quality assurance and maintainability.
ASSESSMENTS	- Initial project proposal and project-suitability review. - Problem-definition, aims and objectives checkpoint. - Teacher conference and initial Analysis tracker.	- Stakeholder evidence and research-analysis review. - Requirements and success-criteria quality check. - Full Analysis milestone and tracker update.	- Analysis improvement check. - Decomposition and module-justification review. - Initial interface-design checkpoint.	- Interface/usability review. - Modelled pseudocode and algorithm-design tasks. - Algorithm coverage checklist and Design milestone review.	- Variables and data-structures table. - Methodology and test-plan review. - Acceptance-test traceability check. - Completed Design milestone.	- Analysis of strong and weak Development entries. - First Development-story entry. - Practical coding/testing checkpoint and evidence review.
FEEDBACK SUPPORTS LEARNING						

	Autumn 1	Autumn 2	Spring 1	Spring 2	Summer 1	Summer 2
	UNIT OF WORK: NEA introduction, project suitability and initial Analysis	UNIT OF WORK: NEA Analysis: investigation, requirements and success criteria	UNIT OF WORK: NEA Design: decomposition, solution structure and initial interface planning	UNIT OF WORK: NEA Design: interface design, usability and algorithm design	UNIT OF WORK: NEA Design: technical choices, methodology and testing plans	UNIT OF WORK: NEA Development: iterative implementation and development evidence
INCLUSION AND ACCESSIBILITY						
SPECIALIST VOCABULARY	NEA, stakeholder, end user, problem definition, aim, objective, scope, complexity, feasibility, constraint, computational solution, decomposition, abstraction	primary research, secondary research, questionnaire, interview, observation, qualitative data, quantitative data, functional requirement, non-functional requirement, usability, accessibility, reliability, success criterion	decomposition, module, subsystem, procedure, function, class, input, process, output, dependency, integration, modular design, traceability	user interface, navigation, consistency, feedback, validation, error prevention, algorithm, pseudocode, sequence, selection, iteration, parameter, return value	variable, constant, attribute, data type, scope, array, list, dictionary, record, file, database, iterative development, prototype, boundary data, acceptance testing	implementation, iteration, milestone, syntax error, logic error, runtime error, debugging, expected outcome, actual outcome, remedial action, retesting, reflection, annotation
QUALITY FIRST TEACHING						

Theory boundary: This map includes the Component 01 and 02 knowledge that directly enables the NEA. Standalone examined content - including processor architecture, operating systems and scheduling, assembly language, number representation, Boolean algebra, compression and the wider networks specification - remains in the separate theory curriculum map, although students may draw on it where their project genuinely requires it.

Specification alignment: OCR A Level Computer Science H446, Version 3.0 (April 2026), Components 01, 02 and 03/04.

Progression to Year 13: continued Development and integration, project-dependent theory application, final system and acceptance testing, stakeholder feedback, Evaluation against requirements and success criteria, maintainability, further improvements, final conclusion and submission.