

DEVELOP COMPUTER SOFTWARE

TVET CDACC UNIT CODE: SEC/OS/CS/CR/03/6/MA

UNIT DESCRIPTION

This unit covers the competencies required to develop computer software. It involves establishing software purpose, analysing software requirements, designing computer software, developing computer software, performing programme testing and maintenance.

ELEMENTS AND PERFORMANCE CRITERIA

ELEMENT	PERFORMANCE CRITERIA
These describe the key outcomes which make up workplace function.	These are assessable statements which specify the required level of performance for each of the elements. <i>(Bold and italicised terms are elaborated in the Range)</i>
1. Identify the fundamentals to programming	1.1 Core programming concepts (variables, data types, operators, control structures) are explained with illustrative examples. 1.2 Problem-solving methods (algorithms, flowcharts, pseudocode) are introduced to support structured development. 1.3 Principles of structured programming (sequence, selection, iteration) are described and applied. 1.4 Modular programming concepts, including functions and procedures, are outlined. 1.5 Syntax and semantics of a chosen programming language are identified and practiced. 1.6 Common error types (syntax, logic, runtime) are explained with examples. 1.7 Coding standards and best practices (naming conventions, comments, readability) are recognized and adopted. 1.8 Fundamentals of debugging and testing are introduced within the development process.

	1.9 The relationship between programming, SDLC, and problem-solving is explained.
2. Design computer software	2.1 Software requirements are analyzed and documented based on user and organizational needs. 2.2 Functional and system specifications are developed to guide software design. 2.3 Program logic is structured using algorithms, flowcharts, and pseudocode. 2.4 Software architecture (e.g., modular, client-server, layered, object-oriented) is selected based on requirements. 2.5 Data structures and databases are designed for efficient storage, retrieval, and processing. 2.6 User interface (UI) and user experience (UX) designs are developed following usability standards. 2.7 Security considerations (e.g., input validation, authentication, encryption, error handling) are incorporated into the design. 2.8 Detailed design documentation is produced to support development and maintenance. 2.9 Design is validated and reviewed with stakeholders for feasibility and alignment with requirements.
3. Develop computer software using an object-oriented programming language (JAVA, C++ or PYTHON)	3.1 OOP concepts (classes, objects, attributes, methods) are applied in development. 3.2 Encapsulation is implemented to protect data and maintain integrity. 3.3 Inheritance is applied to promote code reuse and support hierarchical structures. 3.4 Polymorphism is utilized through method overloading and overriding. 3.5 Abstraction is applied to simplify complexity by

	exposing only essential features. 3.6 Code is written using an appropriate OOP language (e.g., Java, Python, C++, C#). 3.7 Error handling and exception management are implemented for reliable execution. 3.8 Modular, reusable components are developed to support scalability and maintainability. 3.9 Source code is documented in line with coding standards and best practices. 3.10 Developed modules are tested to validate functionality, security, and performance.
4 Perform program testing	4.1 A testing plan is developed in line with requirements and design. 4.2 Unit testing is performed to verify individual modules and functions. 4.3 Integration testing validates interactions between software components. 4.4 System testing ensures the complete solution meets functional and non-functional requirements. 4.5 User acceptance testing (UAT) confirms usability and client satisfaction. 4.6 Test data is prepared to simulate real-world scenarios. 4.7 Bugs, defects, and inconsistencies are documented and communicated. 4.8 Debugging techniques are applied to correct errors. 4.9 Retesting and regression testing are conducted to ensure fixes maintain stability. 4.10 Test outcomes are recorded in reports to support quality assurance and future maintenance.
5 Perform software implementation and maintenance	5.1 An implementation plan is developed outlining deployment procedures, timelines, and responsibilities.

	5.2 Software is installed, configured, and deployed according to standards. 5.3 Data migration and system integration are carried out to ensure smooth transition. 5.4 Post-deployment verification is conducted to confirm installation and functionality. 5.5 User support is provided through training, documentation, and helpdesk services. 5.6 Maintenance activities (patching, updates, version control) are conducted to sustain reliability. 5.7 Performance monitoring tools are used to detect and resolve operational issues. 5.8 Security vulnerabilities are identified and remediated through updates and configurations. 5.9 Corrective, adaptive, and preventive maintenance measures are applied across the lifecycle. 5.10 Implementation and maintenance processes are documented for compliance and continuous improvement. Performance monitoring tools are used to identify and resolve operational issues. 5.11 Security vulnerabilities are regularly assessed and remediated through updates and configuration adjustments. 5.12 Corrective, adaptive, and preventive maintenance measures are applied to sustain software lifecycle. 5.13 Implementation and maintenance activities are documented for auditing, compliance, and continuous improvement.
--	--

RANGE

This section provides work environment and conditions to which the performance criteria apply. It allows for different work environment and situations that will affect performance.

Variable	Range
Problem-solving methods may include but not limited to:	• Algorithms • Flowcharts • Pseudocode
Programming language syntax and semantics may cover but not limited to:	• Language-specific rules • Data handling • Control flow • Semantics applied in practice
Coding standards and best practices may include but not limited to:	• Naming conventions • Code comments • Readability and maintainability
Software requirements analysis may include but not limited to:	• User needs • Organizational needs • Documentation of requirements
UI/UX design may include but not limited to:	• Usability standards • Accessibility • Interface layouts
Security considerations may include but not limited to:	• Input validation • Authentication • Encryption • Error handling
OOP implementation may use but limited to:	• Java • Python • C++
OOP testing may include but not limited to:	• Module validation • Security testing • Performance validation
System testing may cover but not limited to:	• Functional requirements • Non-functional requirements
Implementation planning may include but not limited to:	• Deployment procedures • Timelines

	• Responsibilities
Software deployment may cover:	• Installation • Configuration • Rollout procedures
User support may include but not limited to:	• Training • Documentation • Helpdesk services
Security maintenance may include but not limited to:	• Vulnerability identification • Remediation • Secure configurations
Maintenance measures may include but not limited to:	• Corrective maintenance • Adaptive maintenance • Preventive maintenance

REQUIRED KNOWLEDGE AND UNDERSTANDING

The individual needs to demonstrate knowledge and understanding of:

- Problem-solving methods such as algorithms, flowcharts, and pseudocode to structure program development.
- Language-specific rules, data handling, control flow, and practical application of semantics in programming.
- Coding standards, including naming conventions, use of comments, and techniques for readability and maintainability.
- Analyze and document software requirements based on user needs and organizational objectives.
- Usability standards, accessibility principles, and interface layout design for effective user experiences.
- Secure coding practices such as input validation, authentication, encryption, and error handling.
- Implementation of OOP concepts using languages such as Java, Python, C++.
- Testing methods including module validation, security testing, and performance validation.
- Deployment processes such as installation, configuration, and rollout procedures.

- User support mechanisms including training, documentation, and helpdesk services.

FOUNDATION SKILLS

The individual needs to demonstrate the following foundation skills:

- Communications (verbal and written);
- Troubleshooting
- Problem solving;
- Decision making;
- Planning;
- Report writing;

EVIDENCE GUIDE

This provides advice on assessment and must be read in conjunction with the performance criteria, required knowledge and understanding and range.

1. Critical Aspects of Competency	Assessment requires evidence that the candidate: 1.1 Modular programming concepts, including functions and procedures, are outlined. 1.2 Coding standards and best practices (naming conventions, comments, readability) are recognized and adopted. 1.3 The relationship between programming, SDLC, and problem-solving is explained. 1.4 Software requirements are analyzed and documented based on user and organizational needs. 1.5 Functional and system specifications are developed to guide software design. 1.6 Program logic is structured using algorithms, flowcharts, and pseudocode. 1.7 Software architecture (modular, client-server, layered, object-oriented) is selected based on requirements. 1.8 Data structures and databases are designed for efficient storage, retrieval, and processing. 1.9 Security considerations (input validation, authentication, encryption, error handling) are incorporated into the design.
-----------------------------------	---

	1.10 Detailed design documentation is produced to support development and maintenance. 1.11 Design is validated and reviewed with stakeholders for feasibility and alignment with requirements. 1.12 Inheritance is applied to promote code reuse and support hierarchical structures. 1.13 Source code is documented in line with coding standards and best practices. 1.14 Developed modules are tested to validate functionality, security, and performance. 1.15 A testing plan is developed in line with requirements and design. 1.16 Software is installed, configured, and deployed according to standards. 1.17 Data migration and system integration are carried out to ensure smooth transition. 1.18 Post-deployment verification is conducted to confirm installation and functionality. 1.19 User support is provided through training, documentation, and helpdesk services. 1.20 Maintenance activities (patching, updates, version control) are conducted to sustain reliability. 1.21 Performance monitoring tools are used to detect and resolve operational issues. 1.22 Security vulnerabilities are identified and remediated through updates and configurations.
2. Resource Implications for competence certification	The following resources should be provided: 2.1 Access to relevant workplace where assessment can take place 2.2 Appropriately simulated environment where assessment can take place 2.3 Materials relevant to the proposed activity or tasks
3. Methods of	Competency may be assessed through:

Assessment	3.1 Observation with the help of check list 3.2 Practical demonstrations 3.3 Oral Questioning
4. Context of Assessment	Competency may be assessed individually in the actual workplace or a simulated work place setting
5. Guidance information for assessment	Holistic assessment with other units relevant to the industry sector, workplace and job role is recommended.