Software Quality Assurance Group Assignment Report

Higher National Diploma in Information Technology

ATI –Tangalle

Contents

Software Verification Techniques

-Introduction-
Software verification techniques are essential processes in software development aimed at ensuring the correctness, reliability, and quality of software systems. This report provides an overview of various software verification techniques, including their identification, advantages, disadvantages, rationale for use, examples for syntax verification, limitations, and associated tools.

1. Static Analysis (Syntax Verification)
Identification: Static analysis involves examining code without executing it, focusing on syntax checking to identify language-specific errors.

- **Advantages**

- Early detection of syntax errors.

- Scalability for large codebases.

- Integration into development workflows.

- **Disadvantages**

- Limited to syntax errors; may miss semantic issues.

- Potential for false positives.

- Inability to capture all program behaviors.

- **Why Use:** Ensures compliance with language syntax rules, reducing runtime errors and enhancing code maintainability.

- **Examples for Syntax Verification:**

- Detecting missing semicolons in JavaScript.

- Identifying undeclared variables in Python.

- **Limitations:** May struggle with complex code structures and dynamically generated code.

- **Tools:**

- ESLint (JavaScript)

- Pylint (Python)

- Visual Studio Code (Integrated Development Environment)

2. Dynamic Analysis (Syntax Verification):

- **Identification:** Dynamic analysis, including unit testing and integration testing, indirectly verifies syntax correctness through code execution with various inputs.

- **Advantages:**

- Reveals syntax errors and logic flaws during runtime.

- Provides insights into code behavior.

- Facilitates comprehensive testing.

- **Disadvantages:**

- Limited to detecting syntax errors indirectly.

- Requires creation and maintenance of test suites.

- Time-consuming for complex systems.

- **Why Use:** Complements static analysis by validating syntax correctness through actual execution, aiding in identifying runtime issues.

- **Examples for Syntax Verification:**

-Verifying correct function calls in unit tests.

- Checking input validation in integration tests.

- **Limitations:** May not uncover all possible execution paths and may not simulate real-world scenarios accurately.

- **Tools:**

- JUnit (Java)

- Pytest (Python)

- Jenkins (Continuous Integration)

3. Formal Verification (Syntax Verification):

- **Identification:** Formal verification employs mathematical techniques to prove or disprove the correctness of code, including syntax adherence.

- **Advantages:**

- Guarantees correctness under specified conditions.

- Rigorous verification of syntax and semantics.

- Suitable for critical systems.

- **Disadvantages:**

- Highly resource-intensive.

- Requires expertise in formal methods.

- Limited scalability for large systems.

- **Why Use:** Ensures strict adherence to syntax rules and semantic correctness, crucial in safety-critical applications.

- **Examples for Syntax Verification:**

- Proving correctness of syntax constructs using formal logic.

- **Limitations:** Highly resource-intensive and may not handle real-world uncertainties effectively.

- **Tools:**

- SPIN (Model Checker)

- Isabelle (Theorem Prover)

- Coq (Proof Assistant)

4. Code Reviews (Syntax Verification):

- **Identification:** Manual inspection of code by peers to identify syntax errors, coding style violations, and other issues.

- **Advantages:**

- Finds syntax errors and improves readability.

- Facilitates collaboration and knowledge sharing.

- Enhances code quality through peer feedback.

- **Disadvantages:**

- Subjective assessments based on reviewer expertise.

- Time-consuming for large codebases.

- May not detect all syntax issues.

- **Why Use**: Augments automated techniques by leveraging human judgment to identify syntax errors and enforce coding standards.

- **Examples for Syntax Verification:**

- Spotting syntax errors during code review sessions.

- **Limitations:** Subjective and time-consuming, dependent on reviewer expertise.

- **Tools:**

- GitHub (Code Review Platform)

- Code Climate (Static Code Analysis Tool)

- SonarQube (Code Quality Management Platform)

5. Linters (Syntax Verification):

- **Identification:** Linters analyze source code to flag programming errors, bugs, stylistic errors, including syntax issues.

- **Advantages:**

- Identifies syntax errors and coding style violations.

- Automates syntax issue detection.

- Integrates with development environments for real-time feedback.

- **Disadvantages:**

- Potential for false positives.

- Limited to surface-level syntax checks.

- Requires configuration for custom rule sets.

- **Why Use:** Ensures adherence to coding standards and identifies syntax errors early in development.

- **Examples for Syntax Verification:**

- Enforcing code formatting rules using linters.

- **Limitations:** May produce false positives and have limited depth of analysis.

- **Tools:**

- ESLint (JavaScript)

- RuboCop (Ruby)

- Flake8 (Python)

Group Members

Registration number :-

TAN/IT/2021/F/0019-S.V.P.R Sandamini

TAN/IT/2021/F/0058-M.K.D.Shrivanthika

TAN/IT/2021/F/0068- G.P.H.Nuwandi

TAN/IT/2021/F/0070-D.A.T.H.Kawshani

TAN/IT/2021/F/0105- K.A.N.K.Vijerathna