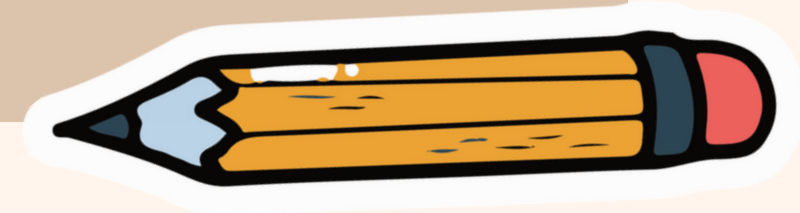


# What is Testing?



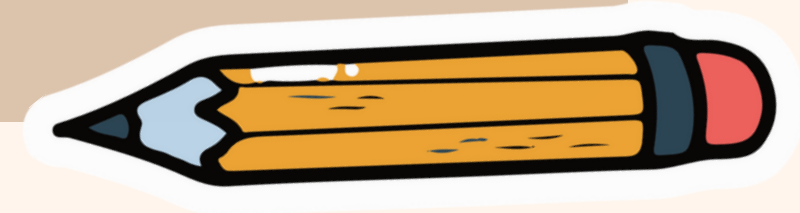
- Most people have had an experience with software that did not work as expected.
- Software that does not work correctly can lead to many problems, including loss of money, time, or business reputation, and even injury or death.
- Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.



# Typical Objectives of Testing



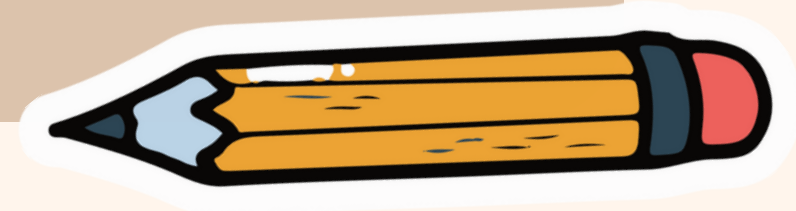
- To prevent defects by evaluate work products such as requirements, user stories, design, and code
- To verify whether all specified requirements have been fulfilled
- To check whether the test object is complete and validate if it works as the users and other stakeholders expect
- To build confidence in the level of quality of the test object
- To find defects and failures by reducing the level of risk of inadequate software quality
- To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object
- To comply with contractual, legal, or regulatory requirements or standards, and/or to verify the test object's compliance with such requirements or standards



# Testing và Debugging



- Testing and debugging are different.
- Executing tests can show failures that are caused by defects in the software
- Debugging is the development activity that finds, analyzes, and fixes such defects.

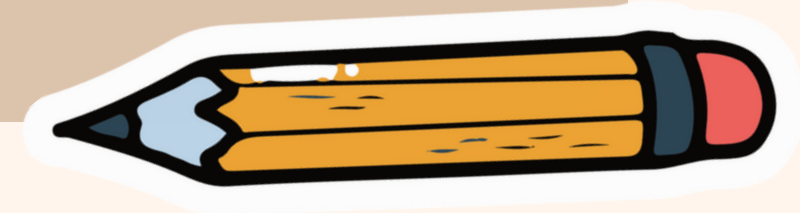


# Errors, Defects, and Failures



A person can make an error (mistake), which can lead to the introduction of a defect (fault or bug) in the software code or in some other related work product.

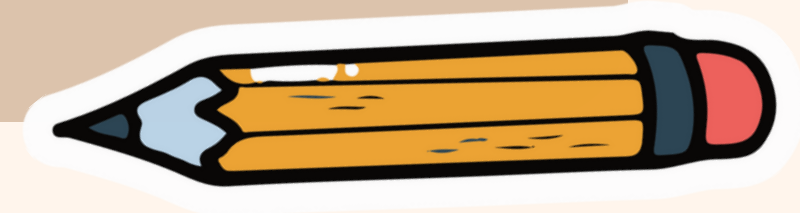
If a defect in the code is executed, this may cause a failure, but not necessarily in all circumstances




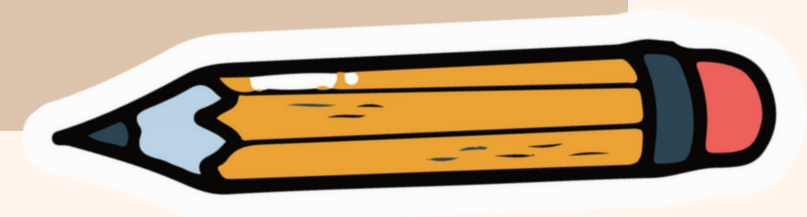
# Testing shows the presence of defects, not their absence




- Testing can show that defects are present, but cannot prove that there are no defects.
- Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, testing is not a proof of correctness.

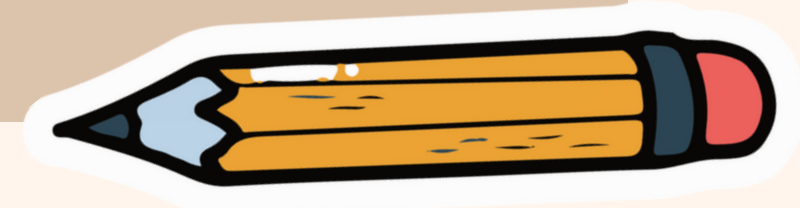


# Exhaustive testing is impossible

- 
- Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases.
  - Rather than attempting to test exhaustively, risk analysis, test techniques, and priorities should be used to focus test efforts.
- 


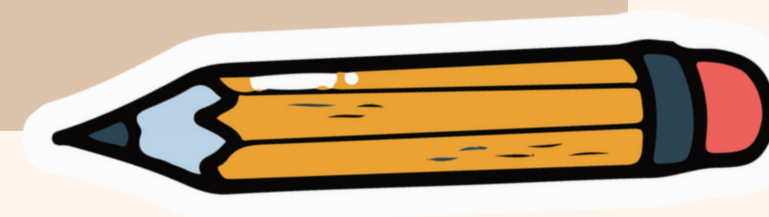
# Early testing saves time and money RAL

- 
- To find defects early, both static and dynamic test activities should be started as early as possible in the software development lifecycle.
  - Early testing is sometimes referred to as shift left.
  - Testing early in the software development lifecycle helps reduce or eliminate costly changes




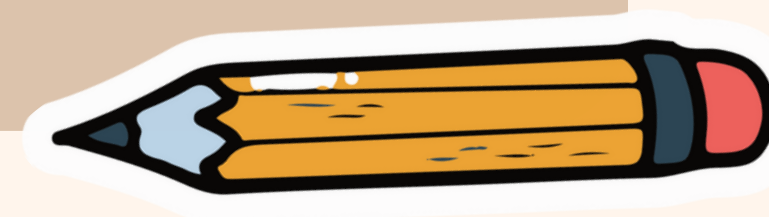


# Defects cluster together

- 
- A small number of modules usually contains most of the defects discovered during pre-release testing or is responsible for most of the operational failures.
  - Predicted defect clusters, and the actual observed defect clusters in test or operation, are an important input into a risk analysis used to focus the test effort
- 

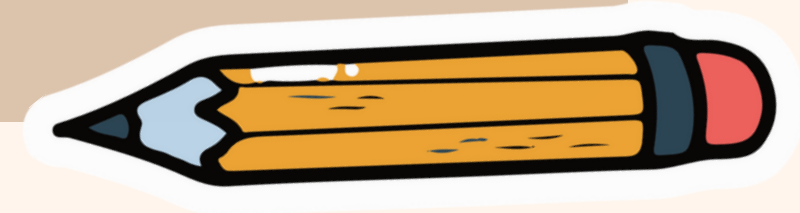


# Beware of the pesticide paradox



- 
- If the same tests are repeated over and over again, eventually these tests no longer find any new defects.
  - To detect new defects, existing tests and test data may need changing, and new tests may need to be written.
  - In some cases, such as automated regression testing, the pesticide paradox has a beneficial outcome, which is the relatively low number of regression defects.
- 

# Testing is context dependent

- Testing is done differently in different contexts
- For example, safety-critical industrial control software is tested differently from an e-commerce mobile app.





# Absence-of-errors is a fallacy

- 
- Some organizations expect that testers can run all possible tests and find all possible defects, but principles 2 and 1, respectively, tell us that this is impossible.
  - Further, it is a fallacy (i.e., a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system.
  - For example, thoroughly testing all specified requirements and fixing all defects found could still produce a system that is difficult to use, that does not fulfill the users' needs and expectations, or that is inferior compared to other competing systems.
- 


# Test Process

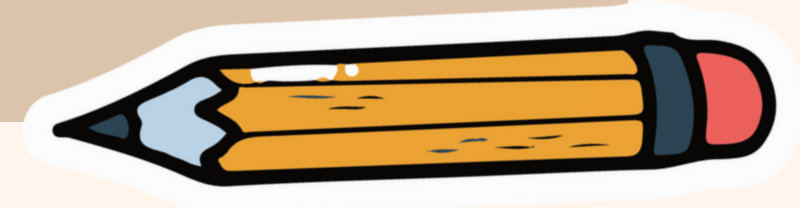
- 
- Test planning
  - Test monitoring and control
  - Test analysis
  - Test design
  - Test implementation
  - Test execution
  - Test completion
- 

# Test Planning



- 
- Test planning involves activities that define the objectives of testing and the approach for meeting test objectives within constraints imposed by the context
  - Test plans may be revisited based on feedback from monitoring and control activities.
- 

# Test monitoring and control

- 
- Test monitoring involves the on-going comparison of actual progress against planned progress using any test monitoring metrics defined in the test plan.
  - Test control involves taking actions necessary to meet the objectives of the test plan (which may be updated over time). Test monitoring and control are supported by the evaluation of exit criteria, which are referred to as the definition of done in some software development lifecycle models





# Test analysis

- 
- Analyzing the test basis appropriate to the test level being considered
  - Evaluating the test basis and test items to identify defects of various types Identifying features and sets of features to be tested
  - Defining and prioritizing test conditions
  - Capturing bi-directional traceability between each element of the test basis and the associated test conditions
- 



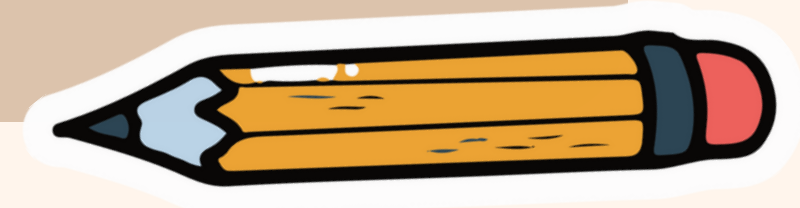
# Test design

- 
- Designing and prioritizing test cases and sets of test cases Identifying necessary test data to support test conditions and test cases
  - Designing the test environment and identifying any required infrastructure and tools
  - Capturing bi-directional traceability between the test basis, test conditions, and test cases
- 

# Test implementation



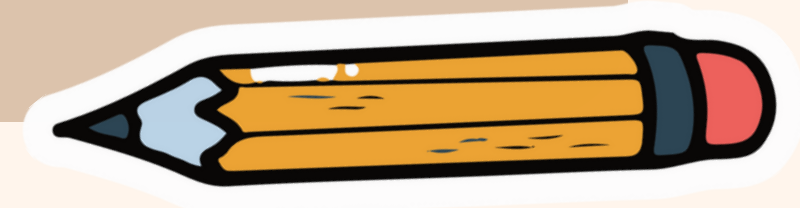
- Developing and prioritizing test procedures, and, potentially, creating automated test scripts
- Creating test suites from the test procedures and (if any) automated test scripts Arranging the test suites within a test execution schedule in a way that results in efficient test execution
- Building the test environment and verifying that everything needed has been set up correctly
- Preparing test data and ensuring it is properly loaded in the test environment
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test suites




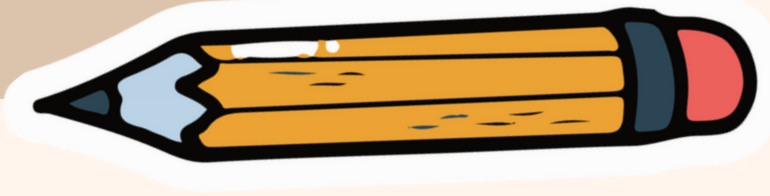
# Test execution



- Recording the IDs and versions of the test item(s) or test object, test tool(s), and testware
- Executing tests either manually or by using test execution tools
- Comparing actual results with expected results
- Analyzing anomalies to establish their likely causes
- Reporting defects based on the failures observed
- Logging the outcome of test execution
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test results.



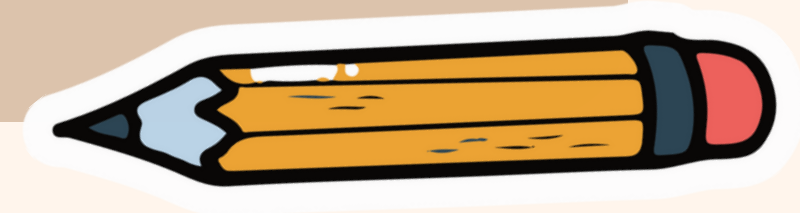
# Test completion

- 
- Checking whether all defect reports are closed, entering change requests or product backlog items for any defects that remain unresolved at the end of test execution
  - Creating a test summary report to be communicated to stakeholders
  - Finalizing and archiving the test environment, the test data, the test infrastructure, and other testware for later reuse
  - Handing over the testware to the maintenance teams, other project teams, and/or other stakeholders who could benefit from its use
  - Analyzing lessons learned from the completed test activities to determine changes needed for future iterations, releases, and projects
  - Using the information gathered to improve test process maturity
- 

# Human Psychology and Testing



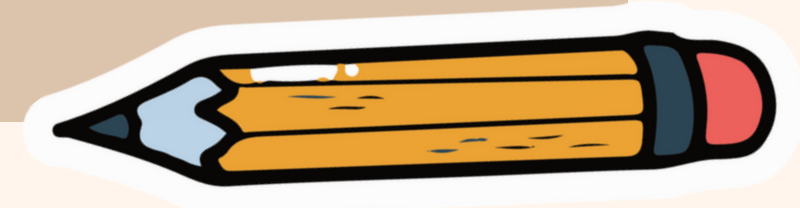
- Start with collaboration rather than battles
- Test monitoring and control
- Emphasize the benefits of testing
- Test design Communicate test results and other findings in a neutral, fact-focused way without criticizing the person who created the defective item.
- Write objective and factual defect reports and review findings.
- Try to understand how the other person feels and the reasons they may react negatively to the information.
- Confirm that the other person has understood what has been said and vice versa.



# Tester's and Developer's Mindsets

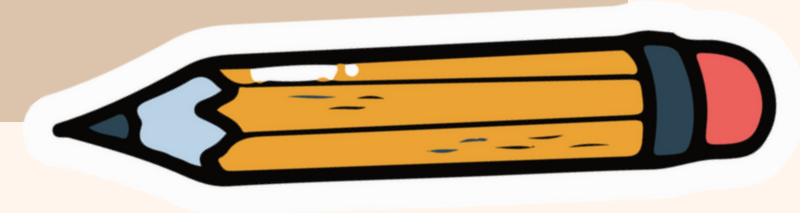


- Developers and testers often think differently.
- The primary objective of development is to design and build a product.
- As discussed earlier, the objectives of testing include verifying and validating the product, finding defects prior to release, and so forth.
- A tester's mindset should include curiosity, professional pessimism, a critical eye, attention to detail, and a motivation for good and positive communications and relationships.
- A developer's mindset may include some of the elements of a tester's mindset, but successful developers are often more interested in designing and building solutions than in contemplating what might be wrong with those solutions.



# Testware

- Testware is created as output work products from the test activities

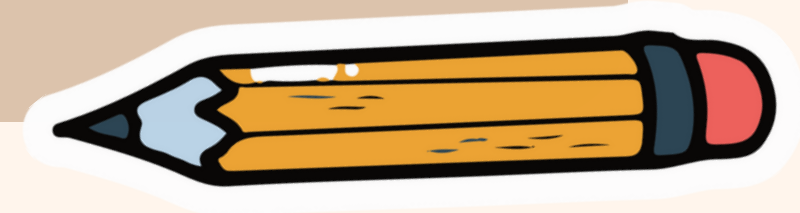




# Test planning work products include



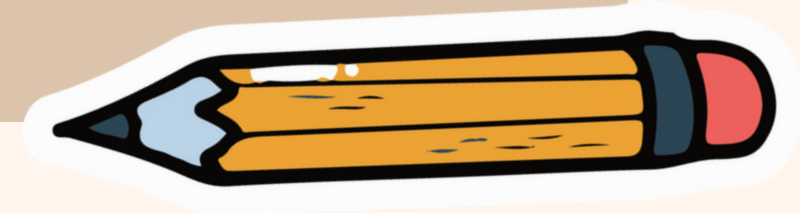
Test plan, test schedule, risk register, and entry and exit criteria Risk register is a list of risks together with risk likelihood, risk impact and information about risk mitigation



# Test monitoring and control work products:



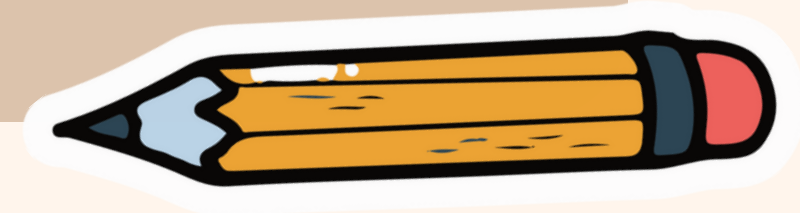
Test progress reports, documentation of control directives and risk information



# Test analysis work products



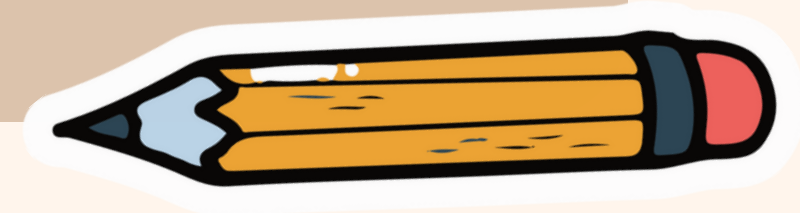
Test analysis work products (prioritized) test conditions (e.g., acceptance criteria), and defect reports regarding defects in the test basis (if not fixed directly).



# Test design work products



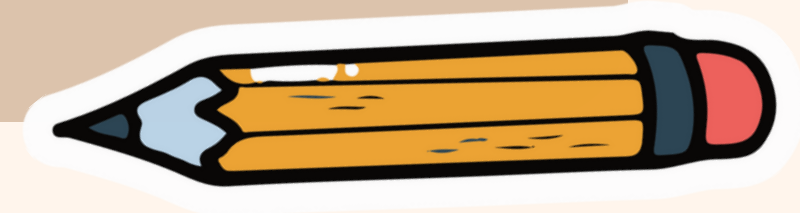
Test design work products (prioritized) test cases, test charters, coverage items, test data requirements and test environment requirements.



# Test implementation work products



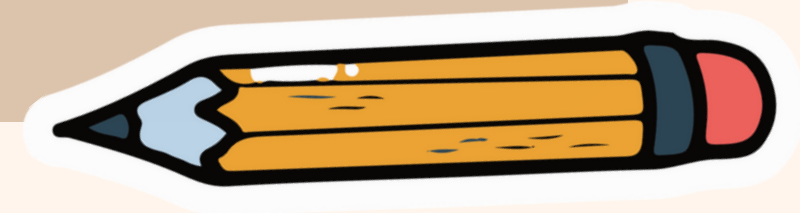
Test procedures, automated test scripts, test suites, test data, test execution schedule, and test environment elements. Examples of test environment elements include: stubs, drivers, simulators, and service virtualizations.



# Test execution work products



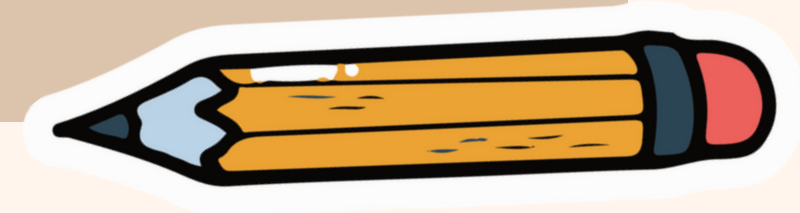
Test completion work products Test execution work products : test logs, and defect reports  
Test completion work products: test completion report, action items for improvement of subsequent projects or iterations, documented lessons learned, and change requests (e.g., as product backlog items).



# Essential Skills





- Thoroughness, carefulness, curiosity, attention to details, being methodical
- Good communication skills, active listening, being a team player
- Analytical thinking, critical thinking, creativity
- Technical knowledge
- Domain knowledge (to be able to understand and to communicate with end users/business representatives)





# Essential Skills

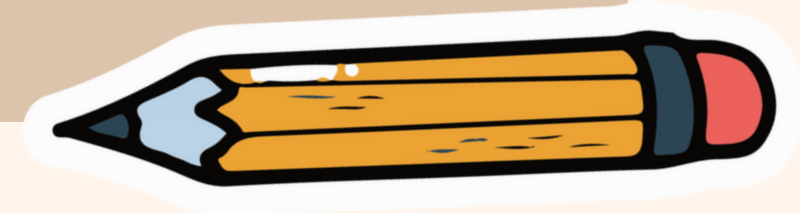
- 
- Analytical thinking, critical thinking, creativity
  - Technical knowledge
  - Domain knowledge (to be able to understand and to communicate with end users/business representatives)
- 

# Test roles



Test roles two principal roles in testing are covered:

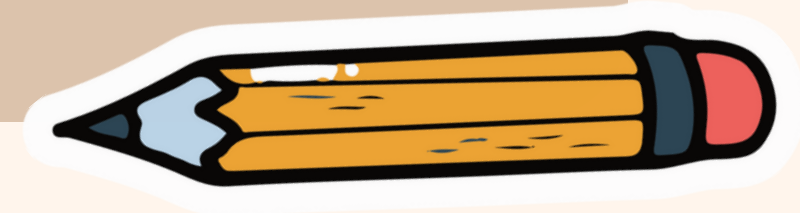
- A test management role
- A testing role



# The test management



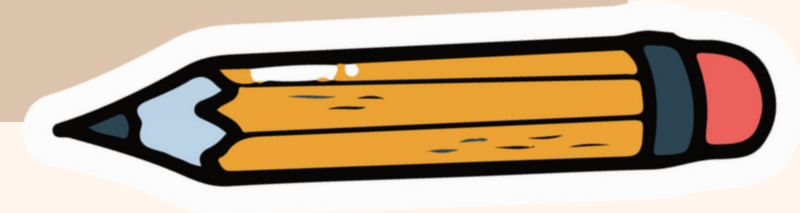
- The test management role takes overall responsibility for the test process, test team and leadership of the test activities.
- The test management role is mainly focused on the activities of test planning, test monitoring and control and test completion.



# The testing



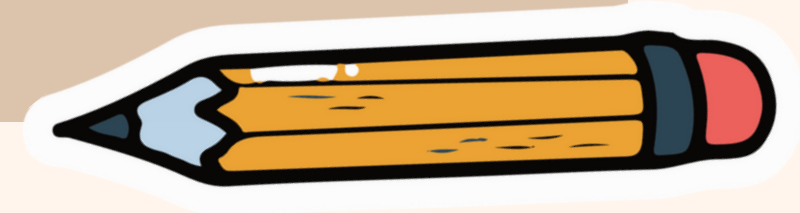
- The testing role takes overall responsibility for the engineering (technical) aspect of testing.
- The testing role is mainly focused on the activities of test analysis, test design, test implementation and test execution.



# Impact of the Software Development Lifecycle



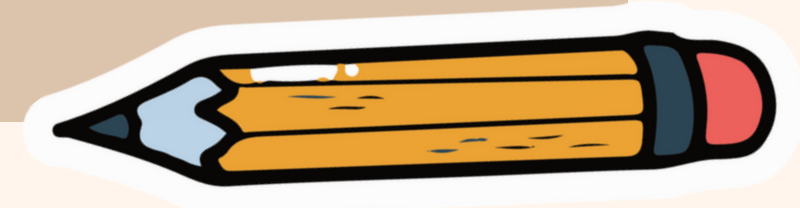
- Scope and timing of test activities (e.g., test levels and test types)
- Level of detail of test documentation
- Choice of test techniques and test approach
- Extent of test automation Role and responsibilities of a tester



# Software Development lifecycle and Good Testing Practices

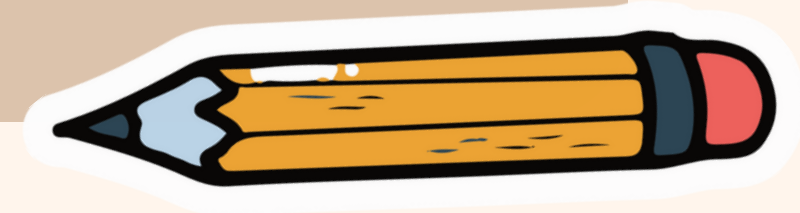


- For every software development activity, there is a corresponding test activity, so that all development activities are subject to quality control.
- Different test levels (see chapter 2.2.1) have specific and different test objectives, which allows for testing to be appropriately comprehensive while avoiding redundancy.
- Test analysis and design for a given test level begins during the corresponding development phase of the SDLC, so that testing can adhere to the principle of early testing.
- Testers are involved in reviewing work products as soon as drafts of this documentation are available, so that this earlier testing and defect detection can support the shift-left strategy.




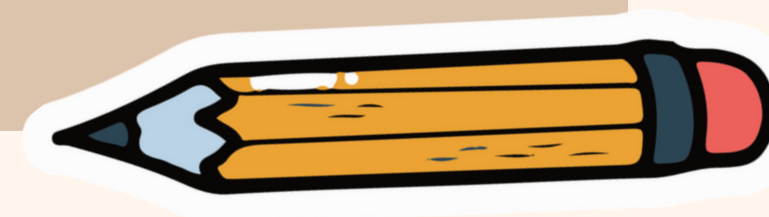
# Test-Driven Development (TDD)

- 
- Directs the coding through test cases
  - Tests are written first, then the code is written to satisfy the tests, and then the tests and code are refactored test cases





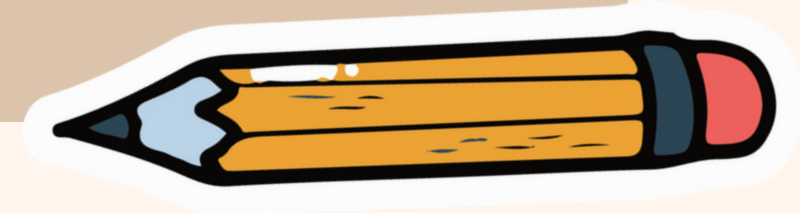
# Acceptance Test-Driven Development (ATDD)

- 
- Derives tests from acceptance criteria as part of the system design process
  - Tests are written before the part of the application is developed to satisfy the tests
- 


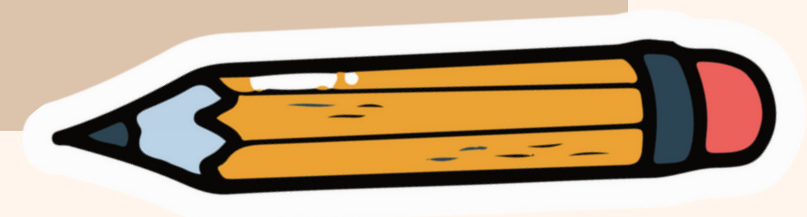
# Behavior-Driven Development (BDD)



- Expresses the desired behavior of an application with test cases written in a simple form of natural language, which is easy to understand by stakeholders - usually using the Given/When/Then format
- Test cases are then automatically translated into executable tests



## CI/CD

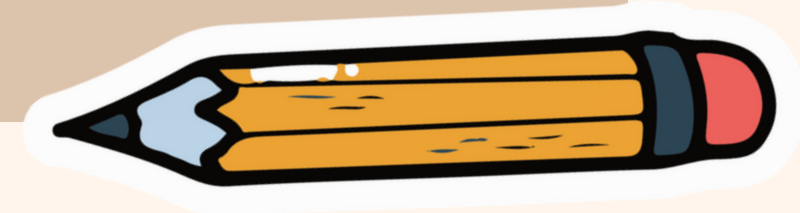
- 
- DevOps promotes team autonomy, fast feedback, integrated toolchains, and technical practices like continuous integration (CI) and continuous delivery (CD).
- 

# Benefits of DevOps



## Benefits of DevOps:

- Fast feedback on the code quality, and whether changes adversely affect existing code CI promotes a shift-left approach in testing by encouraging developers to submit high quality code accompanied by component tests and static analysis
- Promotes automated processes like CI/CD that facilitate establishing stable test environments  
Increases the view on non-functional quality characteristics (e.g., performance, reliability)  
Automation through a delivery pipeline reduces the need for repetitive manual testing
- The risk in regression is minimized due to the scale and range of automated regression tests

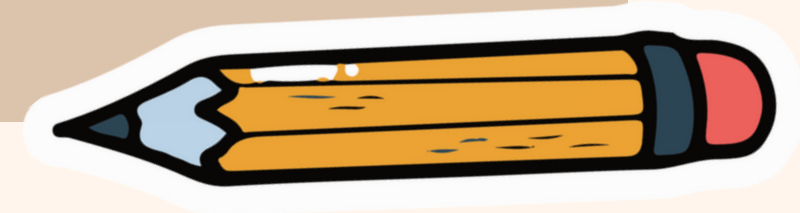


# DevOps is not without its risks and challenges


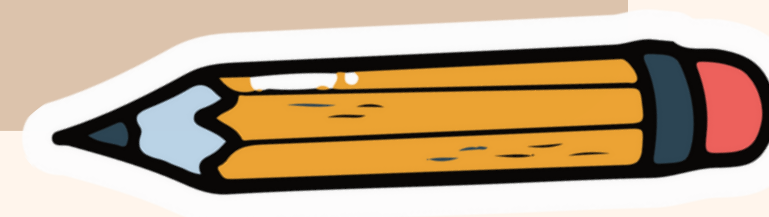


DevOps is not without its risks and challenges, which include:


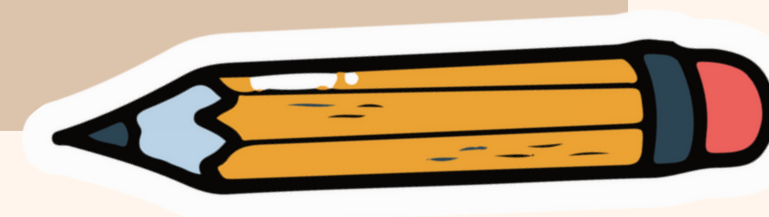
- The DevOps delivery pipeline must be defined and established CI/CD tools must be introduced and maintained
- Test automation requires additional resources and may be difficult to establish and maintain



# Component integration testing

- 
- Component integration testing (also known as unit integration testing) focuses on testing the interfaces and interactions between components.
  - Component integration testing is heavily dependent on the integration strategy approaches like bottom-up, top-down or big-bang.
- 

# System testing

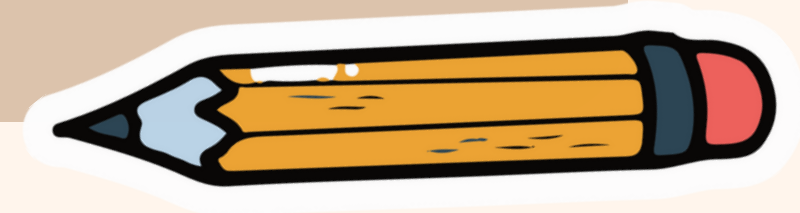
- 
- System testing focuses on the overall behavior and capabilities of an entire system or product, often including functional testing of end-to-end tasks and the non-functional testing of quality characteristics.
  - For some non-functional quality characteristics, it is preferable to test them on a complete system in a representative test environment (e.g., usability).
- 




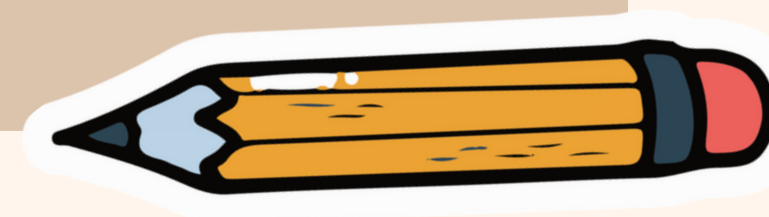
# System integration testing




- System integration testing focuses on testing the interfaces of the system under test and other systems and external services.
- System integration testing requires suitable test environments preferably similar to the operational environment.

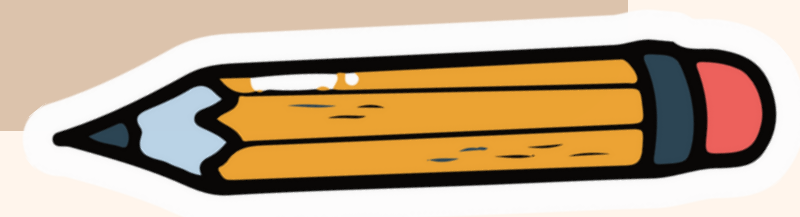


# Acceptance testing


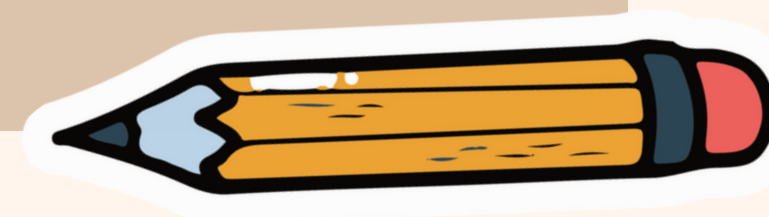
- 
- Acceptance testing focuses on validation and on demonstrating readiness for deployment, which means that the system fulfills the user's business needs. Ideally, acceptance testing should be performed by the intended users.
  - The main forms of acceptance testing are: user acceptance testing (UAT), operational acceptance testing, contractual and regulatory acceptance testing, alpha testing and beta testing,
- 

# Shift-Left Approach

- 
- The principle of early testing is sometimes referred to as shift-left because it is an approach where testing is performed earlier in the SDLC.



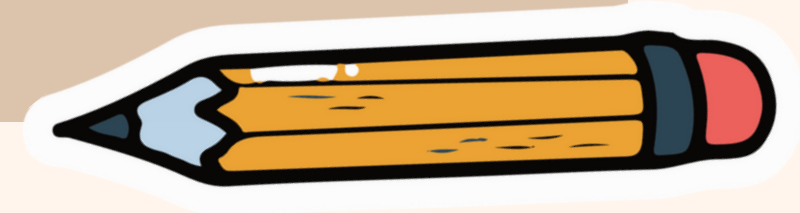
# Retrospectives and Process

- 
- Improvement Retrospectives (also known as “post-project meetings” and project retrospectives) are often held at the end of a project or an iteration, at a release milestone, or can be held when needed.
  - What was successful, and should be retained?
  - What was not successful and could be improved?
  - How to incorporate the improvements and retain the successes in the future?
- 

# Functional testing



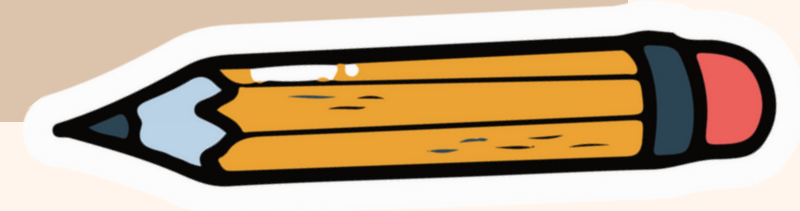
- Functional testing evaluates the functions that a component or system should perform.
- The functions are “what” the test object should do.
- The main objective of functional testing is checking the functional completeness, functional correctness and functional appropriateness.




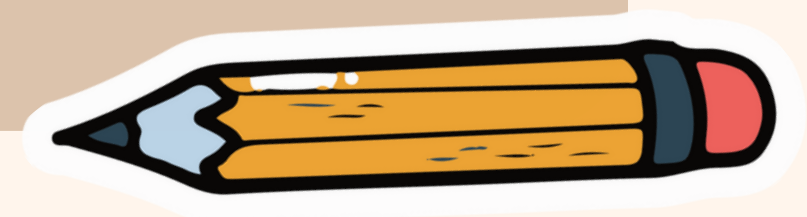
# Non-functional testing



- Non-functional testing evaluates attributes other than functional characteristics of a component or system.
- Non-functional testing is the testing of “how well the system behaves”.
- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability



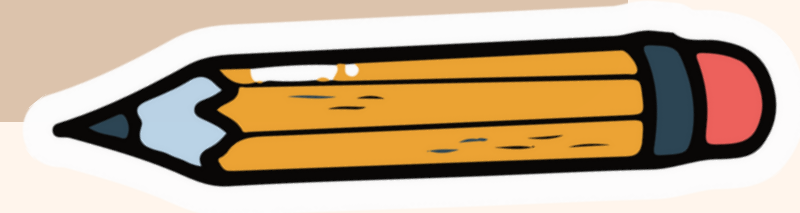
# Black-box testing

- 
- Black-box testing specification-based and derives tests from documentation external to the test object.
  - The main objective of black-box testing is checking the system's behavior against its specifications.
- 

# White-box testing



- White-box testing structure-based and derives tests from the system's implementation or internal structure (e.g., code, architecture, work flows, and data flows).
- The main objective of white-box testing is to cover the underlying structure by the tests to the acceptable level.



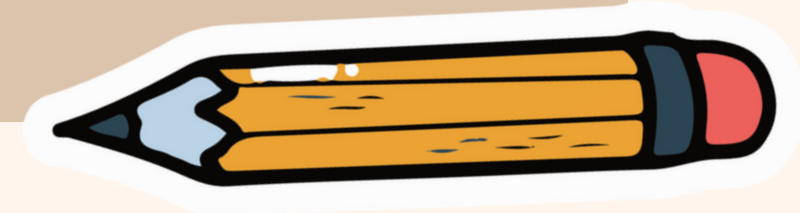


# The scope of maintenance testing



The scope of maintenance testing typically depends on:

- The degree of risk of the change
- The size of the existing system
- The size of the change

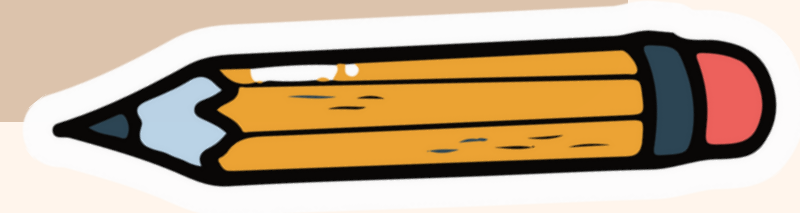


# The triggers for maintenance and maintenance



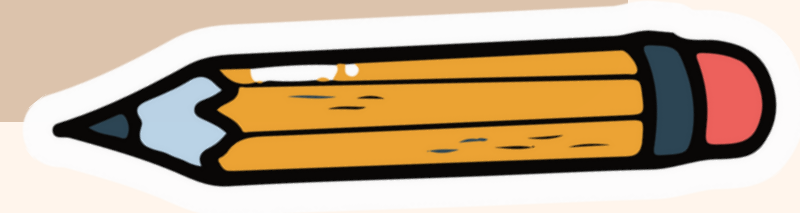
The triggers for maintenance and maintenance testing can be classified as follows:

- Modifications, such as planned enhancements
- Upgrades or migrations of the operational environment
- Retirement, such as when an application reaches the end of its life


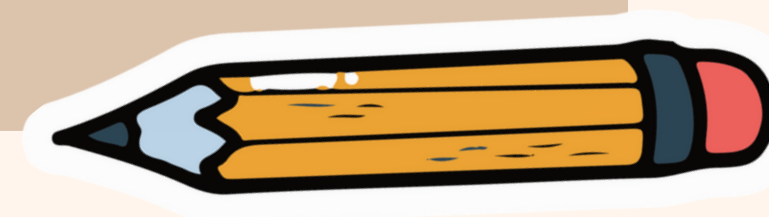


# Static Testing Basics

- In contrast to dynamic testing, in static testing the software under test does not need to be executed.



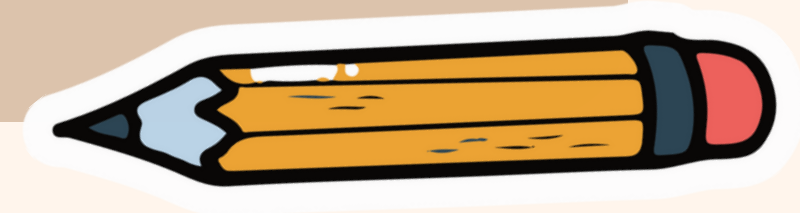
# Work Products Examinable by Static Testing

- 
- Almost any work product can be examined using static testing.
  - Examples include requirement specification documents, source code, test plans, test cases, product backlog items, test charters, project documentation, contracts and models.
- 



# Value of Static Testing Static




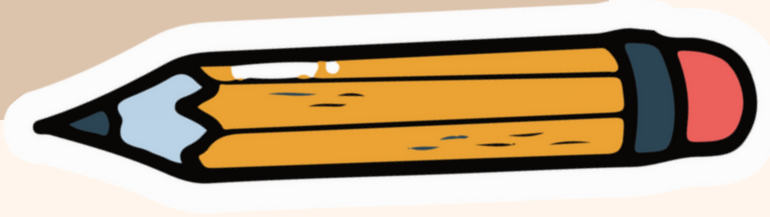
- Value of Static Testing Static testing can detect defects in the earliest phases of the SDLC, fulfilling the principle of early testing.
- Static testing provides the ability to evaluate the quality of, and to build confidence in work products.
- Communication will also be improved between the involved stakeholders



# Differences between Static Testing and Dynamic Testing

- 
- Defects in requirements Design defects
  - Certain types of coding defects
  - Deviations from standards
  - Incorrect interface specifications
  - Specific types of security vulnerabilities
  - Gaps or inaccuracies in test basis coverage
- 

# Benefits of Early and Frequent Stakeholder Feedback

- 
- Early and frequent feedback allows for the early communication of potential quality problems.
  - Frequent stakeholder feedback throughout the SDLC can prevent misunderstandings about requirements and ensure that changes to requirements are understood and implemented earlier.
- 



# Review Process Activities



- 1.Planning: During the planning phase, the scope of the review, which comprises the purpose, the work product to be reviewed, quality characteristics to be evaluated, areas to focus on, exit criteria, supporting information such as standards, effort and the timeframes for the review, shall be defined.
- 2.Review initiation: During review initiation, the goal is to make sure that everyone and everything involved is prepared to start the review. This includes making sure that every participant has access to the work product under review, understands their role and responsibilities and receives everything needed to perform the review.
- 3.Individual review. Every reviewer performs an individual review to assess the quality of the work product under review, and to identify anomalies, recommendations, and questions by applying one or more review techniques (e.g., checklist-based reviewing, scenario-based reviewing).
- 4.Communication and analysis. Since the anomalies identified during a review are not necessarily defects, all these anomalies need to be analyzed and discussed.
- 5.Fixing and reporting: For every defect, a defect report should be created so that corrective actions can be followed- up. Once the exit criteria are reached, the work product can be accepted. The review results are reported.

