Exercise 9 :

Task 1: Explain how the number of defects remaining in your software at delivery affects the product support. Draw a chart illustrating the correlation.

Defects remaining at delivery refer to bugs, errors, or issues that are still present in the software when it is handed over to users or customers.

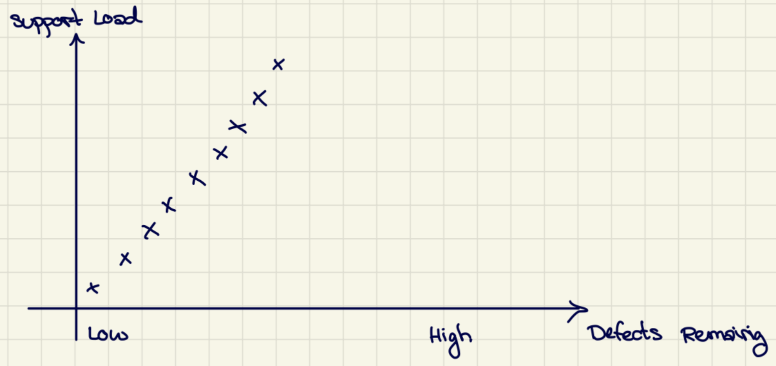
Impact on product support:

1. **Increased Support Requests:**
   More defects usually lead to more users experiencing problems, which increases the number of support tickets and calls.
2. **Longer Resolution Time:**
   Complex or numerous defects can take longer to diagnose and fix, prolonging the time support teams spend on each case.
3. **Higher Support Costs:**
   More defects mean more resources spent on handling issues—this increases the overall cost of support.
4. **Reduced Customer Satisfaction:**
   When users face many bugs, their satisfaction drops, potentially harming the product's reputation and future sales.

Exercise 9 Task 1:

Chart Correlation between Number of Defects Remaining at Delivery and Product Support Load

- X-Achse : Number of Defects Remaining at Delivery (from low to high)
- Y-Achse : Product Support Load (includes volume of support tickets, time, spent, and costs)
- Curve: The curve rises sharply as defects increase, meaning support load increases exponentially with more defects.



- At low defect levels, support load is minimal, fewer issues mean fewer calls and faster resolutions.
- As defects increase, the support load grows rapidly, reflecting the strain on resources and customer dissatisfaction.

Task 2: Give 5 arguments for and against developers testing their own programmes.

Arguments For developers testing their own programs:

1. **Deep knowledge of the code**
   Developers understand the design, logic, and edge cases better than anyone, enabling them to create precise and effective tests.
2. **Faster feedback**
   Developers can catch bugs immediately during coding, shortening the feedback loop and reducing time spent fixing issues later.
3. **Cost-effective**
   Early testing by developers reduces the reliance on dedicated testers for basic bugs, cutting costs and freeing testers for higher-level tasks.
4. **Improves code quality**
   Knowing they will test their own work encourages developers to write cleaner, more maintainable code from the start.
5. **Continuous integration readiness**
   Automated unit tests written by developers integrate well into CI/CD pipelines, supporting frequent releases with fewer regressions.

Arguments Against developers testing their own programs:

1. **Bias and blind spots**
   Developers may unconsciously avoid testing scenarios where they assume the code will work, missing important bugs.
2. **Lack of independence**
   Without an objective perspective, developers might overlook requirements misunderstandings or usability issues.
3. **Tunnel vision**
   Developers focus on *how* the program is built, while testers focus on *what* the program should do — skipping the latter can lead to gaps.
4. **Limited test coverage mindset**
   Developers often write tests for what the code is supposed to do, but testers excel at thinking of what could go wrong or be misused.
5. **Time constraints**
   Balancing feature development and thorough testing can overburden developers, leading to rushed or incomplete testing.

Task 3: in Intellij

Task 4: What is regression testing ?

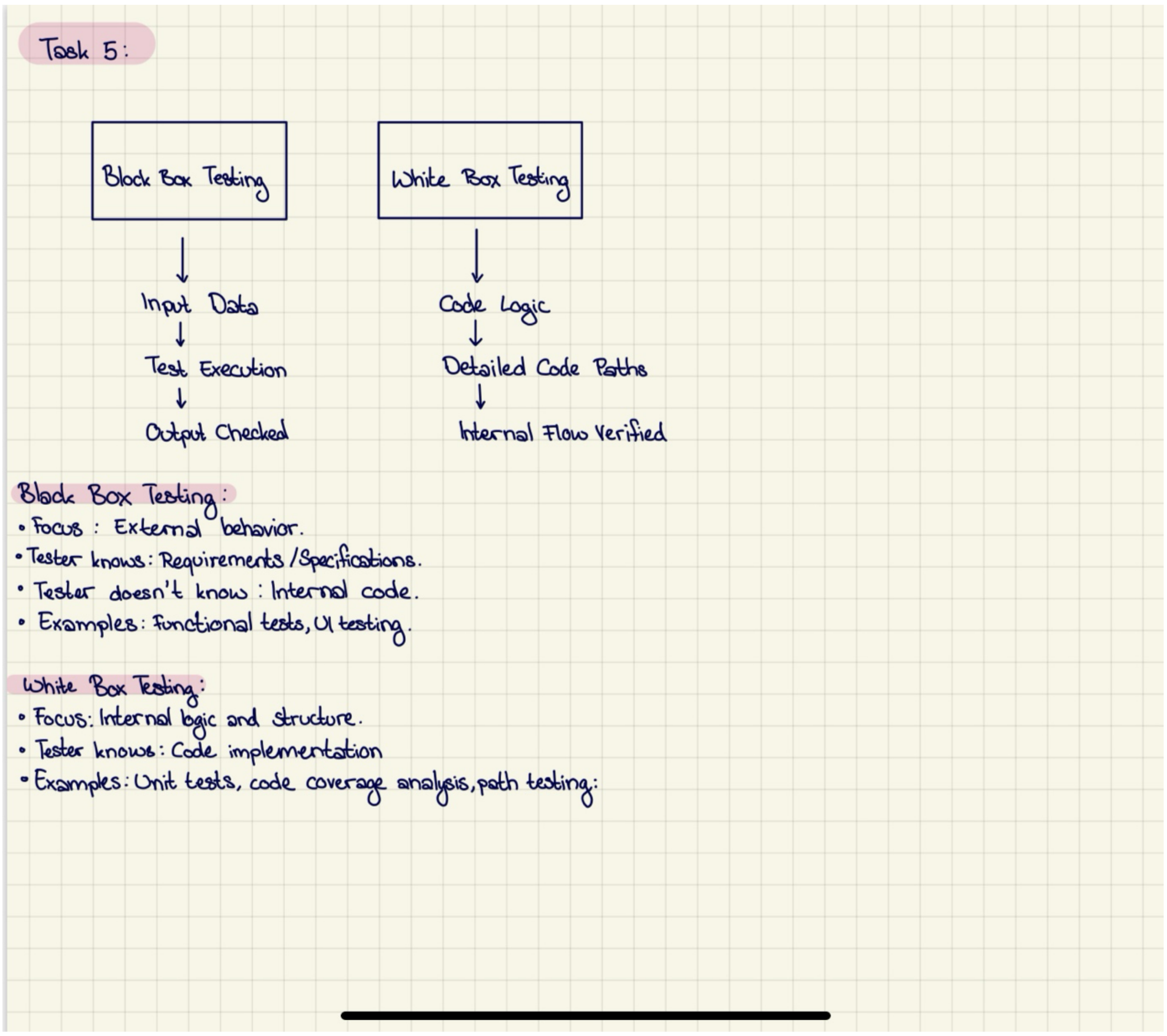
Regression testing is the process of **retesting** software after changes (like bug fixes or feature updates**)** to ensure that previously working functionality hasn't been broken.

Purpose:

- Catch unintended side effects from changes.
- Ensure new code doesn't break old code.

Example: If a new feature is added to a login system, regression tests would rerun all existing login tests to ensure old functionality still works as expected.

Task 5: Draw an illustration of the term black box testing and white box testing and describe the difference.



Task 5:

$$\begin{tikzcd}
\boxed{\text{Black Box Testing}} \arrow[d] & \boxed{\text{White Box Testing}} \arrow[d] \\
\text{Input Data} \arrow[d] & \text{Code Logic} \arrow[d] \\
\text{Test Execution} \arrow[d] & \text{Detailed Code Paths} \arrow[d] \\
\text{Output Checked} & \text{Internal Flow Verified}
\end{tikzcd}$$

Black Box Testing:
- Focus : External behavior.
- Tester knows: Requirements / Specifications.
- Tester doesn't know : Internal code.
- Examples: functional tests, UI testing.

White Box Testing:
- Focus: Internal logic and structure.
- Tester knows: Code implementation
- Examples: Unit tests, code coverage analysis, path testing.