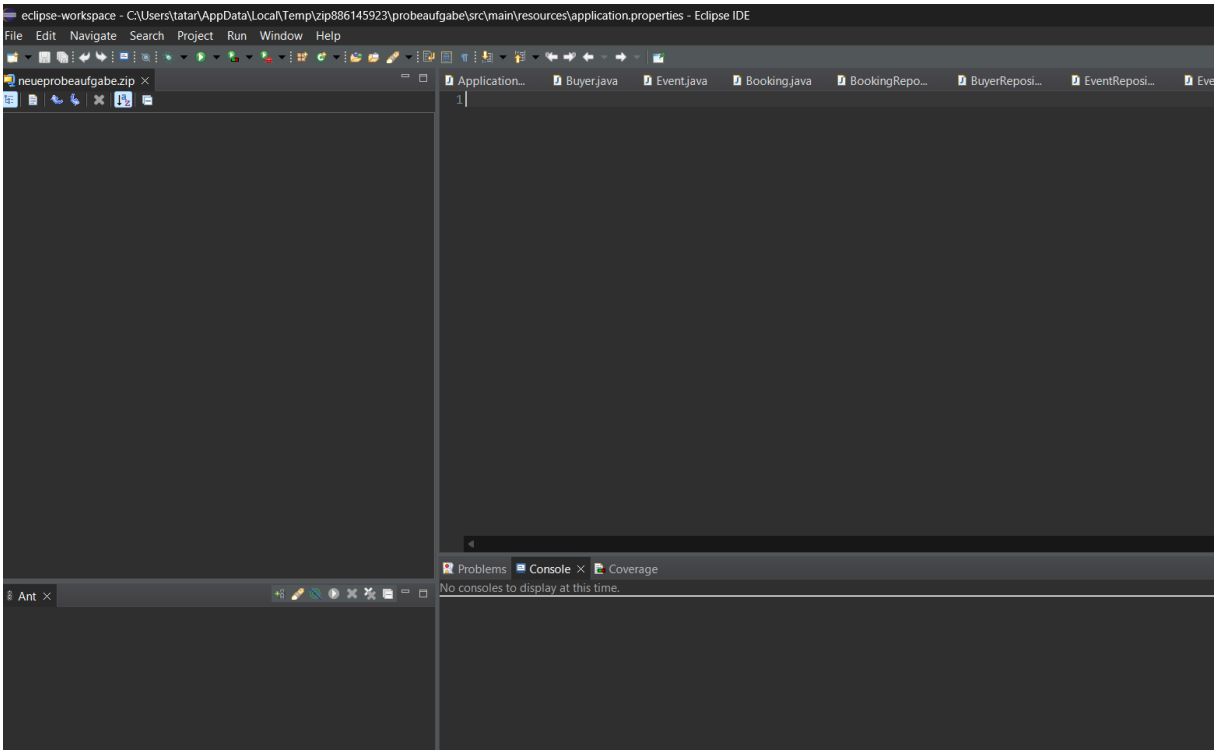
1.Aufgabe

- **Code Completion**: IDEs suggest code while you type, speeding up development.
- **Syntax Highlighting**: Different parts of the code are color-coded for better readability.
- **Debugging Tools**: Built-in debuggers help find and fix errors easily.
- **Error Detection**: Many IDEs show errors in real-time as you write the code.
- **Project Management**: IDEs help organize and manage large projects efficiently.
- **Integrated Version Control**: Many IDEs support Git, SVN, etc., directly inside the environment.
- **Build and Run Tools**: You can compile and run your code without leaving the IDE.
- **Refactoring Support**: Easier to rename, move, or restructure code safely.
- **Plugins and Extensions**: Extend functionality easily with plugins.
- **Multiple Language Support**: Many IDEs support several programming languages in one place.



2.Aufgabe

This problem is related to the famous **Collatz Conjecture** (also known as the 3n + 1 problem), which remains unproven in general.
However, for numbers less than **1 million**, it has been **tested and verified** by computer experiments.
Here's a structured explanation:

1. **Definition**:
   The function repeatedly applies two simple rules based on whether nnn is even or odd.
2. **Goal**:
   Show that starting with any n<1,000,000n < 1,000,000n<1,000,000, the sequence eventually reaches 1.
3. **Empirical Evidence**:
   Computer calculations have been performed up to very large numbers (well beyond 1 million), and all tested numbers so far eventually reach 1.
4. **Behavior**:
   - If nnn is even, n/2n/2n/2 is smaller, so nnn decreases.
   - If nnn is odd, 3n+13n+13n+1 becomes even, and the next division by 2 reduces the number.
5. **Known Facts**:
   - Many sequences rise and fall, but all known tested sequences eventually drop down to 1.
   - For n<1,000,000n < 1,000,000n<1,000,000, the termination has been verified.
6. **Conclusion**:Even though there is no formal proof for all natural numbers, **for all n<1,000,000n < 1,000,000n<1,000,000, the Ulam function terminates at 1**.

6.Aufgabe

Software Design:
Software design is the process of **defining the architecture, components, modules, interfaces, and data** for a system to satisfy specified requirements.
It focuses on **how** the system will be built.

Difference from Software Analysis:

- **Software Analysis** identifies **what** the system must do by gathering requirements and understanding problems.
- **Software Design** describes **how** to implement these requirements as a working system.

In short:

- Analysis = "What is needed?"

- Design = "How will it be built?"

7.Aufgabe

Why is Software Design Necessary?
Software design is necessary because it provides a **clear blueprint** for developers, ensuring that the system's components fit together correctly.
It helps avoid confusion, reduces errors, and ensures that the project meets user requirements efficiently and systematically.

Project Without Design?
Imagine a project started without any planning — developers would begin coding without coordination.

Consequences:

- **Conflicting components** that don't work together.
- **Increased bugs** and technical debt.
- **Higher development costs** due to rework.
- **Missed deadlines** because of unexpected issues.
- **Poor scalability** and maintainability of the software.

8.Aufgabe

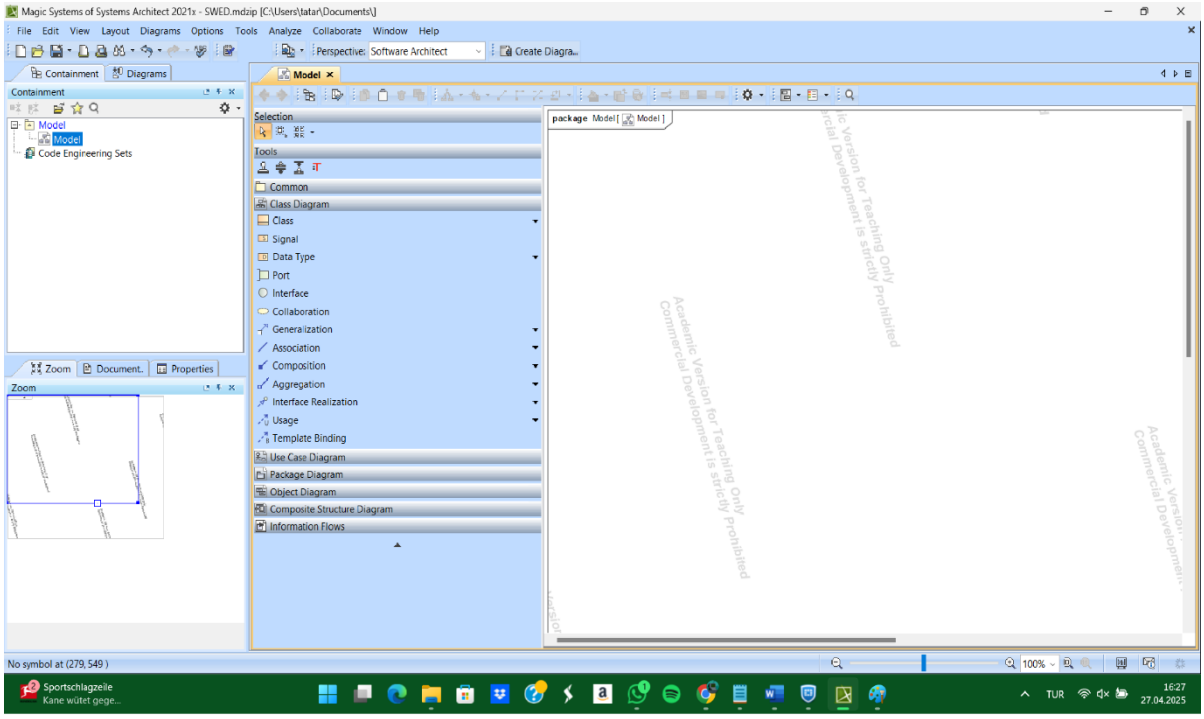
Are they Independent or Interdependent?
They are **interdependent**. Each design activity influences and depends on the others.

Example Explanation:
Imagine building an **online shopping website**:

- **Architectural Design** decides that the system will have separate modules for product management, ordering, and payments.
- **Database Design** must store product information, user data, and order history according to that architecture.
- **User Interface Design** must allow users to view products, place orders, and track payments — all relying on the database and the architectural flow.
- **Component Design** must create individual elements (like shopping cart, product detail component) that interact with both UI and backend systems.

Summary:
If the database changes, the UI might need adjustments. If the architecture shifts, components must be restructured. Therefore, **they are strongly interdependent**.

4.Aufgabe



3.Aufgabe