在 WinForms 应用中，实现分层架构主要是通过将代码组织到不同的项目（或命名空间）中，并定义它们之间的引用关系和通信方式。

1. 用户界面层（UI Layer）在 WinForms 中的实现

- **职责**：这是用户直接看到和操作的部分。它负责显示数据、接收用户输入（比如点击按钮、输入文字），并将这些操作传递给业务逻辑层。
- **WinForms 实现方式**：
  - **项目结构**：通常是一个独立的 WinForms 项目，包含所有的窗体（Form）、用户控件（UserControl）等。
  - **组件**：例如，MainForm.cs 可能显示测试进度，ConfigForm.cs 用于配置参数。
  - **交互**：当用户点击一个按钮时，UI 层会捕获这个事件，然后调用业务逻辑层中相应的方法。UI 层**不直接**进行复杂的业务计算或数据库操作。它只负责“展示”和“收集”信息。
  - **数据绑定**：可以使用 WinForms 的数据绑定功能，将 UI 控件直接绑定到业务逻辑层提供的数据对象上，实现数据的自动更新。

2. 业务逻辑层（Business Logic Layer - BLL）在 WinForms 中的实现

- **职责**：这是整个软件的“大脑”，处理所有的核心业务规则和逻辑。它接收 UI 层的请求，进行计算、验证数据，并协调数据访问层来获取或保存数据。
- **WinForms 实现方式**：
  - **项目结构**：通常是一个独立的类库（Class Library）项目。
  - **组件**：包含各种业务逻辑类，例如 TestManager.cs（负责测试流程控制）、DeviceScheduler.cs（负责设备调度）、DataProcessor.cs（负责数据处理）。
  - **交互**：UI 层会调用 BLL 中的公共方法。BLL 会根据业务需求，调用数据访问层（DAL）来获取或保存数据，然后将处理结果返回给 UI 层。BLL 是 UI 和 DAL 之间的桥梁。
  - **业务规则**：所有与业务相关的规则（例如“测试步骤必须按顺序执行”、“设备状态必须是空闲才能开始测试”）都应该在这里实现。

3. 数据访问层（Data Access Layer - DAL）在 WinForms 中的实现

- **职责**：这是负责与数据库进行交互的部分。它知道如何连接数据库、执行 SQL 查询、保存数据等。它将数据库的细节封装起来，业务逻辑层不需要知道具体的数据库类型或操作方式。
- **WinForms 实现方式**：
  - **项目结构**：通常是另一个独立的类库（Class Library）项目。
  - **组件**：包含数据访问类，例如 TestConfigRepository.cs（用于访问测试配置数据）、TestDataRepository.cs（用于访问测试结果数据）。

- **交互**：BLL 会调用 DAL 中的方法来获取或保存数据。DAL 负责将业务对象（例如一个 TestConfiguration 对象）转换为数据库可以理解的格式（例如 SQL 语句），并执行操作。它将数据库操作的结果（例如查询到的数据）再转换回业务对象，返回给 BLL。
- **数据库技术**：根据文档，可以使用 SQLite 或 Access。DAL 会包含连接这些数据库的代码，以及执行增删改查（CRUD）操作的方法。

4. 插件层（Plugin Layer）在 WinForms 中的实现

- **职责**：这是为了支持设备驱动和功能扩展而设计的。它允许系统在不修改核心代码的情况下，通过加载外部插件来增加新的功能或支持新的设备。
- **WinForms 实现方式**：
  - **项目结构**：可以是一个或多个独立的类库项目，每个项目代表一个插件。
  - **组件**：每个插件会实现预定义的接口（例如 IDevicePlugin），这个接口定义了插件必须提供的方法（如 Start、Stop、GetStatus）。
  - **核心平台（Core Platform）**：在 WinForms 应用中，核心平台（通常是 BLL 的一部分或一个独立的管理模块）会负责加载这些插件。它会扫描特定文件夹，找到符合接口的 DLL 文件，并动态加载它们。
  - **交互**：BLL 会通过插件接口来调用插件的功能，而不需要知道具体是哪个设备或哪个插件在执行操作。

总结 WinForms 中的分层实现：

- **项目划分**：在 Visual Studio 中，您会创建多个项目：一个 WinForms 应用程序项目（UI 层），以及多个类库项目（BLL、DAL、Plugin Layer）。
- **引用关系**：
  - **UI 层项目会引用 BLL 项目。**
    - **详细解释**：这意味着 UI 层（例如您的 WinForms 窗体）能够“看到”并调用 BLL（业务逻辑层）中定义的公共类和方法。UI 层需要 BLL 来处理用户操作（比如点击“开始测试”按钮），因为 UI 层本身不包含复杂的业务逻辑。这种引用是单向的，BLL 不能直接引用 UI 层，从而保证了 UI 层可以独立于业务逻辑进行更改。
  - **BLL 项目会引用 DAL 项目。**
    - **详细解释**：BLL（业务逻辑层）需要与数据进行交互，但它不直接操作数据库。因此，BLL 会引用 DAL（数据访问层）项目，以便调用 DAL 中封装的数据操作方法（例如保存配置、读取测试结果）。同样，这种引用是单向的，DAL 不能直接引用 BLL，这确保了 DAL 只专注于数据操作，不依赖于任何特定的业务逻辑。
  - **BLL 项目（或核心平台模块）会动态加载插件层项目。**

        - **详细解释**：与前两种直接引用不同，插件层通常采用“动态加载”的方式。这意味着 BLL 在编译时并不知道具体有哪些插件，而是在运行时根据需要加载插件（例如，扫描特定文件夹中的 DLL 文件）。核心平台会通过预定义的接口（如 IDevicePlugin）来与这些动态加载的插件进行交互。这种方式使得系统具有极高的扩展性，无需重新编译主应用程序即可添加新的设备支持或功能模块。
    - **DAL 项目通常不引用其他业务层或 UI 层项目，因为它只负责数据操作。**
        - **详细解释**：DAL 的设计原则是保持其独立性。它只负责与数据库的通信，不应该依赖于任何上层（BLL 或 UI）的逻辑。这样，如果将来需要更换数据库类型（例如从 SQLite 切换到 SQL Server），只需要修改 DAL 层，而不会影响到 BLL 和 UI 层。
- **通信**：层与层之间通过接口或公共方法进行通信，避免直接访问其他层的内部实现细节。