

Lab Assignment

ARCHITECTURAL ISSUES

SHAHZEB | SP22-BSE-073 | DEC 31,2025

# CHOOSEN SOFTWARES BY ME

Ma na windows vista ma identify kara ha driver compatibility issue ko.

### **Windows Vista**

* **Flaw**: **Driver Compatibility Issue**
* **Fix**: Microsoft worked with hardware manufacturers to update drivers, provided a compatibility mode for legacy drivers, and expanded the Windows Update platform to distribute updated drivers. They also improved developer tools to help manufacturers create compatible drivers.

### **Lessons Learned**

The Windows Vista driver compatibility problem underscored the importance of:

1. **Backward Compatibility**: Maintaining some level of compatibility with older drivers to ease the transition.
2. **Vendor Collaboration**: Working closely with hardware manufacturers from the early stages of OS development.
3. **Clear Communication**: Explaining architectural changes to users and developers early to prepare them for the transition.

### **2. Windows XP**

* **Flaw**: **Security Vulnerabilities**
* **Fix**: Microsoft introduced **Service Pack 2 (SP2)**, which enhanced security by introducing features like a built-in **firewall**, **Data Execution Prevention (DEP)**, and **Windows Update** for automatic security patch installation. They also improved user account controls to reduce exposure to threats.

### **3. Google Chrome**

* **Flaw**: **Memory Usage Issues**
* **Fix**: Google implemented the **multi-process architecture** to separate tabs into different processes, isolating them from one another to improve stability and reduce memory consumption. They also optimized memory management through various updates, leading to better resource utilization.

### **4. iOS (iPhone OS)**

* **Flaw**: **App Compatibility and Performance Issues**
* **Fix**: Apple made changes by introducing **App Thinning** and **64-bit support** to ensure apps ran efficiently on different device models. They also released new developer tools and improved backwards compatibility to ensure smooth app performance across various iOS versions.

### **5. Windows 95**

* **Flaw**: **System Instability and Software Crashes**
* **Fix**: Microsoft implemented the **Windows 95 OSR2 (OEM Service Release 2)** update, which included system stability improvements, better memory management, and a more reliable file system (FAT32). This update also brought enhanced hardware support and a more stable user interface.

### **Lessons Learned**

The Windows Vista driver compatibility problem underscored the importance of:

1. **Backward Compatibility**: Maintaining some level of compatibility with older drivers to ease the transition.
2. **Vendor Collaboration**: Working closely with hardware manufacturers from the early stages of OS development.
3. **Clear Communication**: Explaining architectural changes to users and developers early to prepare them for the transition.

**My Solution:**

**SAVE THIS CODE IN SEPARATE CLASS AND DEVICE MANAGER CLASS CODE IN ANOTHER CLASS**

**from abc import ABC, abstractmethod**

**# Define an abstract base class for the Driver**

**class Driver(ABC):**

**@abstractmethod**

**def initialize(self):**

**pass**

**@abstractmethod**

**def get\_device\_status(self):**

**pass**

**@abstractmethod**

**def send\_data(self, data):**

**pass**

**# Old driver that uses legacy APIs**

**class OldDriver(Driver):**

**def initialize(self):**

**print("Initializing old driver using legacy methods...")**

**def get\_device\_status(self):**

**return "Old Driver Status"**

**def send\_data(self, data):**

**print(f"Sending data via old driver: {data}")**

**# New driver that uses modern APIs**

**class NewDriver(Driver):**

**def initialize(self):**

**print("Initializing new driver using modern methods...")**

**def get\_device\_status(self):**

**return "New Driver Status"**

**def send\_data(self, data):**

**print(f"Sending data via new driver: {data}")**

**#### Adapter for backward compatibility (Old Driver Wrapper)**

**class OldDriverAdapter(Driver):**

**"""**

**Adapter class to ensure backward compatibility with the old driver.**

**It wraps the old driver in a way that it can be used with the new system.**

**"""**

**def \_\_init\_\_(self, old\_driver: OldDriver):**

**self.old\_driver = old\_driver**

**def initialize(self):**

**# Use legacy initialization method**

**print("Using adapter to initialize the old driver...")**

**self.old\_driver.initialize()**

**def get\_device\_status(self):**

**# Use the old driver to get the status**

**return self.old\_driver.get\_device\_status()**

**def send\_data(self, data):**

**# Use the old driver to send data**

**self.old\_driver.send\_data(data)**

**CLASS DIVICE MANAGER**

class DeviceManager:

def \_\_init\_\_(self, driver: Driver):

self.driver = driver

self.driver.initialize()

def check\_device\_status(self):

status = self.driver.get\_device\_status()

print(f"Device Status: {status}")

def send\_device\_data(self, data):

self.driver.send\_data(data)

# Function to detect driver version (Simulated)

def detect\_driver\_version(driver\_type: str):

if driver\_type == "old":

return OldDriver()

elif driver\_type == "new":

return NewDriver()

else:

raise ValueError("Unknown driver type!")

# Simulate software running with backward compatibility for old drivers

def main():

# Simulate detecting an old driver

driver\_type = "old" # This could be dynamic in real-world applications

driver = detect\_driver\_version(driver\_type)

# If old driver, wrap it with an adapter for backward compatibility

if isinstance(driver, OldDriver):

driver = OldDriverAdapter(driver)

# Initialize the device manager with the compatible driver

device\_manager = DeviceManager(driver)

# Check device status and send data

device\_manager.check\_device\_status()

device\_manager.send\_device\_data("Test Data")

# Now, let's simulate running with a new driver

driver\_type = "new"

driver = detect\_driver\_version(driver\_type)

# Initialize the device manager with the new driver

device\_manager = DeviceManager(driver)

# Check device status and send data

device\_manager.check\_device\_status()

device\_manager.send\_device\_data("Test Data")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT:**

**Using adapter to initialize the old driver...**

**Initializing old driver using legacy methods...**

**Device Status: Old Driver Status**

**Sending data via old driver: Test Data**

**Initializing new driver using modern methods...**

**Device Status: New Driver Status**

**Sending data via new driver: Test Data**