Absolutely! Here's your **complete SVV Final Exam Study Notes** again — this time with each original question included directly before the answer section, along with **expanded explanations** and **clear formatting** for copy-paste into your own revision document.

✅ BITP 3253: Software Verification and Validation – Final Exam Notes (with Questions)

🟩 Question 1: Static Analysis and Reliability in a Smart Vending Machine

📝 Original Question:

Your team is tasked with reviewing the software module of a smart vending machine installed on campus. This machine allows users to select snacks and pay using a linked e-wallet account via QR scan or NFC tap.

Despite successful payments being deducted from users' accounts, several reports indicate:

- Items are not dispensed.
- The machine freezes if idle for too long.
- Maintenance errors are not always detected, leading to confusion and lost money.

As the QA engineer, you're required to perform a static code analysis and conduct reviews to improve reliability and traceability of the system before the software is re-deployed.

a) Identify THREE possible code quality issues that a static analysis tool might flag in such a system.
b) Discuss how early reviews and static analysis can reduce customer complaints in embedded systems.
c) Construct a decision table for the vending eligibility scenario:

The user's e-wallet balance must be equal to or greater than the item price, the selected item must be in stock, and the machine must not be undergoing maintenance. Only when all three conditions are satisfied is vending permitted.

✅ a) Static Code Issues a Tool Might Flag

1. **Null Pointer Dereference**
    - A critical issue where a program tries to use a memory address that hasn't been set. This can crash the vending software, especially when interacting with hardware like dispensers.
2. **Uncaught Exceptions or Missing Error Handling**

- For example, if the dispenser motor fails and the error isn't caught, the machine might freeze or fail silently.

3. **Dead Code or Unreachable Code**
    - Code that will never execute may exist due to faulty logic, and may hide bugs or inflate system complexity.
4. **Memory Leaks**
    - Especially dangerous in embedded systems where memory is limited. Could result in system hang or slow response.
5. **Magic Numbers / Poor Naming**
    - Hardcoded values (e.g., if (x == 3)) with no explanation reduce maintainability and cause confusion.
6. **Resource Not Released (e.g., Files, Locks)**
    - May lead to system freezing or inconsistency when trying to access files or hardware components again.

✅ **b) Benefits of Static Analysis + Early Review**

1. **Early Detection of Bugs**
    - Identifies defects before code runs, saving time and cost in debugging later.
2. **Improves System Stability**
    - Helps identify race conditions, hardware conflicts, or memory mismanagement common in real-time embedded systems.
3. **Improves Maintainability**
    - Forces use of best practices, modular design, and clean documentation, which reduces future errors.
4. **Increases Customer Satisfaction**
    - Fewer bugs = fewer complaints and less downtime.
5. **Enhances Traceability**
    - Helps map defects back to requirements or design stages.

✅ **c) Decision Table – Vending Eligibility**

Sufficient Balance	Item In Stock	Not Under Maintenance	Allow Vending	Message
Yes	Yes	Yes	✅ Yes	None
Yes	Yes	No	❌ No	Maintenance Mode
Yes	No	Yes	❌ No	Item Out of Stock
Yes	No	No	❌ No	Out of Stock + Maintenance
No	Yes	Yes	❌ No	Insufficient Balance
No	Yes	No	❌ No	Balance + Maintenance Error
No	No	Yes	❌ No	Balance + Out of Stock
No	No	No	❌ No	All Conditions Failed

🟩 Question 2: Use Case Test – Advisor Booking Feature

📝 Original Question:

The university recently launched a new feature in its smart campus mobile application called "Request Personal Academic Advisor Session".

This feature allows students to:

- Log in with secure credentials
- View assigned advisors
- Select a topic (course, internship, etc.)
- Pick a time slot
- Submit the request
- Receive notifications (confirmed/rescheduled)
- Access session details in dashboard

System Requirements:

- No double-booking

- Time zones handled correctly
- Only valid students can book
- All actions logged

👉 Create a **use case test case** for this scenario.

✅ **Use Case Test Case: Book Academic Advisor Session**

Field	Content
Test Case ID	TC_UC_001
Use Case Name	Book Academic Advisor Session
Actor	Valid Student
Description	Student books session with advisor via mobile app
Preconditions	Student must be logged in with a valid account
Test Data	Topic: Course Planning, Slot: 10AM, Advisor: Dr. Lee
Basic Flow	Login → View advisor list → Select topic → Pick slot → Submit
Postconditions	Booking created, and notification sent
Alternate Flow	If time slot is booked → Suggest alternate time
Expected Result	Confirmation shown on dashboard with reminder

---

🟩 **Question 3: Software Quality Models in Medical Diagnosis Software**

📝 **Original Question:**

You're working on a medical app using machine learning for diagnosis.

a) Define software quality and how implicit/explicit requirements affect QA.
b) Define the McCall quality model.
c) Define the ISO/IEC 25010 model.
d) Explain FOUR McCall attributes with examples in your app.
e) Explain difference between Defect Management and Quality Attribute Approach.

✅ a) Software Quality + Implicit/Explicit Requirements

- **Software Quality**: The extent to which software meets functional and non-functional requirements.
- **Explicit Requirements**: Stated in SRS (e.g., feature must detect cancer from images).
- **Implicit Requirements**: Assumed by users (e.g., it must not crash or expose personal data).

✅ b) McCall's Quality Model

Three Categories:

1. **Product Operation**: Correctness, usability, reliability, integrity, efficiency
2. **Product Revision**: Maintainability, flexibility, testability
3. **Product Transition**: Portability, reusability, interoperability

---

✅ c) ISO/IEC 25010 Model

Eight Quality Attributes:

- Functional suitability
- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

✅ d) 4 McCall Attributes – Explained with Examples

Attribute	Description	Medical App Example
Correctness	Meets software specifications	Provides accurate diagnosis suggestions

Attribute	Description	Medical App Example
Integrity	Ensures security and data protection	Protects patient data with encryption
Efficiency	Uses minimal CPU, memory	Fast ML response even on low-end devices
Testability	Easy to isolate and validate components	Test ML model independently from the UI layer

---

✅ e) Defect Mgmt vs Quality Attribute Approach

Aspect	Defect Management Approach	Quality Attribute Approach
Timing	After development	During design and planning
Focus	Finding and fixing bugs	Building quality in from the start
Tools	Bug tracking tools	Requirement models, standards

🟩 Question 4: Test Management – Government Tax Filing System

📝 Original Question:

You are the Test Manager for a government tax filing system.
A system upgrade is scheduled during peak filing season. You must plan testing, control risks, and track progress.

a) Identify and explain FOUR responsibilities of a test manager.
b) Define entry and exit criteria. Give TWO examples for each.
c) Name THREE metrics to help track this project.
d) Explain TWO common testing risks and suggest mitigation strategies.

✅ a) Responsibilities of a Test Manager

1. **Test Planning**
    - Define objectives, scope, resources, schedule, and risk-based priorities.
2. **Team Management**
    - Assign roles, coordinate test activities, and handle conflicts.

3. **Risk Management**
    - Identify risks early (e.g., peak usage time), and define mitigation actions.
4. **Progress Monitoring & Reporting**
    - Use metrics and status reports to track testing effectiveness.

✅ b) Entry and Exit Criteria (Examples for Tax System)

Entry Criteria	Exit Criteria
Code freeze completed	All severity 1 and 2 bugs are resolved
Test environment configured	95% of planned tests executed successfully
Test cases reviewed and approved	No open critical issues
Test data is prepared and validated	Acceptance criteria fully met

---

✅ c) Three Useful Metrics

1. **Test Case Execution Rate**
    - % of executed test cases out of total planned.
    - Helps monitor progress.
2. **Defect Density**
    - Number of bugs per KLOC (1,000 lines of code).
    - Tracks code quality.
3. **Defect Leakage Rate**
    - % of bugs found after release.
    - Indicates testing thoroughness.

✅ d) Common Risks and Mitigation

Risk	Mitigation Strategy
Unclear or changing requirements	Early communication and static requirement reviews

Risk	Mitigation Strategy
Tight deadlines during peak season	Prioritize high-risk features; automate regression tests

---

🟩 Question 5: Tool Support – Multi-Platform E-Commerce Testing

📝 Original Question:

You're leading testing for an e-commerce platform (web + mobile + backend).

a) List and explain FOUR types of testing tools.
b) Give THREE benefits and TWO risks of using tools in large-scale projects.
c) What are THREE actions to take before introducing a new testing tool? Justify them.

✅ a) Four Types of Testing Tools

Tool Type	Function	Examples
Test Management	Organize test cases, traceability, and planning	TestRail, Zephyr
Automation Testing	Execute repeatable tests quickly and reliably	Selenium, Appium
Defect Tracking	Log, assign, and track bugs	Jira, Bugzilla
Performance Testing	Measure response time, stress capacity	JMeter, LoadRunner

---

✅ b) Benefits and Risks

Benefits:

1. **Faster execution** – Run tests quickly and more frequently.
2. **Improved repeatability** – Same tests run under consistent conditions.
3. **Better documentation** – Auto-generated logs and reports.

Risks:

1. **High learning curve** – New tools may require time to train the team.
2. **Over-dependence on automation** – Might miss exploratory or usability issues.

✅ c) Three Pre-Implementation Actions

1. **Needs Assessment**
    - Identify specific challenges (e.g., large test suite, multi-platform) and choose tools that solve them.
2. **Pilot Evaluation**
    - Test the tool on a smaller module before scaling across the project.
3. **Training and Onboarding**
    - Conduct team workshops to ensure efficient use of the tool.

🟩 Question 6: CFG & Cyclomatic Complexity – Delivery App

📝 Original Question:

Below is a simplified pseudocode for a smart delivery system that determines shipping method by weight:

cpp

CopyEdit

```
1.  int main() {
2.      int weight;
3.      string method;
4.      cout << "Enter the weight (in kg): ";
5.      cin >> weight;
6.      if(weight <= 2){
7.          method = "Standard Mail";
8.      }
9.      else if(weight > 2 && weight <= 20){
10.         method = "Courier Service";
11.     }
12.     else {
13.         method = "Freight Shipping";
14.     }
```

```
15.     cout << "Shipping Method: " << method;
16.     return 0;
17. }
```

a) Draw the control flow graph (CFG).
b) Calculate the Cyclomatic Complexity.

✅ a) Control Flow Graph (CFG)

Here's a simplified breakdown of the control flow:

```
css
CopyEdit
[Start]
  ↓
[Input Weight]
  ↓
[weight <= 2?]
 /     \
Y       N
↓       ↓
Standard  [weight > 2 && <= 20?]
          /       \
         Y          N
         ↓          ↓
     Courier       Freight
         \         /
        [Print Method]
           ↓
         [End]
```

✅ **b) Cyclomatic Complexity**

Formula:
V(G) = E - N + 2P
Where:

- E = Number of edges = 8
- N = Number of nodes = 7
- P = Number of connected components (usually 1)

Answer:
V(G) = 8 - 7 + 2 = 3
✅ So, minimum **3 test cases** are needed to fully test all paths.