MODULE 5 EXERCISES

Exercise I

- What are the classes B and C regarding IEC 62304?

  Class B and Class C are two of the three software safety classes defined in IEC 62304 (the standard for medical device software lifecycle). The classification is based on the severity of harm that could result from a software failure:

  A. Class A: No injury is possible from software failures (lowest risk).
  B. Class B: A software failure could cause non-serious injury to the patient or user. Non-serious injury means any injury that is not life-threatening and is reversible or minor (e.g., temporary discomfort).
  C. Class C: A software failure could lead to serious injury or death of a patient or user. Serious injury has a specific definition (typically something that is life-threatening, or results in permanent impairment, etc.).

  In practice, if a piece of software is Class B or C, it means it's safety-critical. IEC 62304 requires progressively more rigorous processes for Class B and even more for Class C. For design specifically, Class B and C software must have a documented architecture and detailed design (whereas Class A can sometimes get by with just high-level design or even none explicitly if extremely simple). Also, testing, risk management, and documentation efforts are increased for Class B and C.

Exercise II

- How would you evaluate the quality of a SOUP?

  Evaluating SOUP quality is about performing due diligence: treat the external software as part of your system that needs to meet certain requirements and be risk-managed. You look at manufacturer info, plan risk control measures (like monitoring for updates, sandboxing it if needed, etc.), and ensure it doesn't undermine your system's safety or effectiveness. For high-risk cases, you might even consider alternatives if a SOUP seems too risky (e.g., use a more proven library, or even develop it in-house if necessary).

  To answer this, let's break down what evaluating a SOUP entails:

  A. Identify the SOUP clearly: Know exactly what software component you're dealing with – its name, version, publisher, and what functionality it provides.
  B. Understand its intended use vs. your intended use: Does the SOUP do what you need, and is it intended for that kind of use?
  C. Assess the SOUP's provenance and development rigor: Since by definition SOUP is of unknown provenance (or not developed under your process), find out what you can about its quality:
     1) Does it have documentation? (A well-documented component is easier to evaluate and integrate.)
     2) Is it widely used and tested in industry? (E.g., Flask is widely used – a good sign – whereas a random GitHub project with 5 stars might be risky).
     3) What is the reputation of its developer community or company?
     4) Are there any certifications or prior use in medical devices? (Sometimes companies use certified versions of libraries or those approved by regulatory guidance.)
  D. Review known issues (anomaly lists) and risks: Check the SOUP's issue tracker or release notes for known bugs, especially any that could impact safety or security.

E. Define validation and testing for the SOUP: You should plan to test the software component in the context of your system. For example, write integration tests that specifically exercise the SOUP component as you use it.
F. Documentation and Traceability: Document all the above in your design history. Keep a record of the SOUP evaluation.

Exercise Software Requirements Specification I

- Evaluate the TMS (Training Management System) SRS document to answer the following questions:
  1. Who created the final version?
     Osamah Yacoub and **Ahmad Arrabi** finalized version 1.0 of the SRS document on **Dec 31, 2002.**
  2. What is the intended use?
     The SRS is intended to define **software requirements** for the **E-Government Training Management System (TMS)**. It will be used by the client for approval and by the development team for **design, implementation, and testing** purposes.
  3. Who are the intended users?
     The system is designed for the following user groups (actors):
     1) **System Administrator**
     2) **PMU Administrator** (Project Management Unit)
     3) **TP Administrator** (Training Provider)
     4) **Training Coordinator (TC)** (Government entities)
     5) **GoJ Employee** (Government of Jordan employee)
  4. What is the standard environment?
     From the **Assumptions and Dependencies** section, the environment includes:
     1) **Web browser**: Microsoft Internet Explorer 5.0 or higher
     2) **Languages**: English and Arabic (Arabic for trainees and localized data)
     3) **Screen resolution**: 800x600
     4) **Runtime environment**: Provided and maintained by the client (AlliedSoft not responsible)
     5) **Web-based system** using **3-tier architecture with XML support**
  5. Pick a functional requirement and check if that requirement is
     Selected Requirement: FR01 - Course/Test Maintenance
     Description: Add, update, and delete courses or tests.
     Evaluation:
     1) **Complete**: Yes, includes creation, update, delete operations, plus exceptional cases.
     2) **Accurate**: Yes, clearly maps to user needs and roles (PMU Administrator).
     3) **Unambiguous**: Yes, flow and use-case structure make it specific.
     4) **Traceable**: Yes, traceable in the use case section **4.3.2 Course/Test Maintenance**.
     5) **Testable**: Yes, each CRUD operation and its outcomes (including errors) are testable.
- Is there a requirement regarding data security? If yes, how does that look like?
  Yes. Section **5.2 Security Requirements** defines **user security levels**:
  1) **PMU**: Full control
  2) **TP Administrators**: Limited to center information and test/course data
  3) **Training Coordinators**: Limited to employee nominations
  4) **GoJ Employees**: Can view personal data and submit evaluations

- Is there a requirement how the software should be designed? If yes, how does that look like?
  Yes. Section **5.1 System Requirements** and **3 High-Level System Architecture** specify:
  1) The system must use a **Web-based interface**
  2) Must follow a **3-tier architecture**
  3) Must support **XML for interoperability** with other e-Government applications