4DV608 – Advanced Software Design

- *Assignment 0*

Author: Sarpreet Singh Buttar
Email: Sb223ce@student.lnu.se

Task 1

Impact: Software design plays an important role to avoid faults in dependable and secure computing. The paper specifically mentioned that a software design defect may lead to a failure in the system. In addition, an error can be also seen as a flaw in the software design. Furthermore, fault avoidance imposes discipline and conditions impacts on the software design. For instance, software designers are prohibited to make complex designs. Moreover, fault avoidance also enables software designer to model their designs. The model can be used to predict the behavior of the respective design. On the other hand, fault tolerance also impacts the software design. It divides the design in to two categories named as robustness and redundant. Robustness design deals with unpredicted inputs, changing condition of the environment in which the external model operates, etc. Whereas, redundant design handles various types of redundancy such as information, time, physical and model.

Following are the three core problems including their possible mitigation:

1. **Development Faults:** are the faults that are introduced in the system by the environment in which it operates during the development phase. Users, infrastructure, administrators, etc., are the examples of the environment elements.

   Possible Mitigations:
   - Verify whether the system meets the desired properties
   - Validate whether the system fulfills the given specifications

2. **Interaction Faults:** are the faults that are introduced in the system by the environment in which it operates during the use phase. Users, physical world, intruders, providers, etc., are the examples of the environment elements.

   Possible Mitigations:
   - Apply corrective maintenance to expel faults that have formed one or more reported errors
   - Apply preventive maintenance to find and remove faults before they even appear and reported during runtime

3. **Service Failure:** is an event in which a service fails to behave according to its specifications. Such events are unwanted and can only be identified once it occurred.

   Possible Mitigations:
   - Do qualitative or ordinal evaluation to categorize, determine, and rank the failure events in order to prevent system failure
   - Do quantitative or probabilistic evaluation to estimate which properties of the system will satisfy in the event of a failure

Task 2

Reflection: Software faults are the most common reasons in the failure of a computer system. Aging related bugs is one of the types of software faults cause many issues, e.g., high failure rate, low performance, that can crash or hang the system. Software aging is caused by the effects of various active aging related bugs. Therefore, it leads to various problems in the system such as memory leaks, non-terminated threads, unreleased locks, etc. In safety-critical systems, these problems can even take a human life. *Software rejuvenation* is a proactive approach that enables the system to deal with software aging by executing three periodical steps: terminate an application, clean the internal state of the application, and restart the application. These periodical steps help the system to recover its performance. Approaches such as Non-Accelerated and Accelerated testing are commonly used to study software aging. Both approaches track the indicators of software aging in order to understand its behavior. The indicators are measurable and therefore, state-of-the-art contains various techniques that combines accelerated testing and machine learning to predict the optimal time to start rejuvenation. Hence, these techniques increase the availability of the system and reduce the cost and time.

Two software aging problems that I recognized:

1. **Memory Leaks:** refers to a situation in which data present in the memory is no longer used but still holding the memory. It is one of the major software aging problem. I faced this problem in a database course. My Java program throws *OutOfMemoryError* while reading data from various files.

   Workaround:
   To overcome this problem, close the stream after reading each file. It enables the stream to release the memory when no longer required.

2. **Unreleased Locks:** represents a situation in which a thread locks a system's resource but fails to unlock it and leads to deadlock. I encountered this problem in an operating system course in which my Java program never ends at the desired stage.

   Workaround:
   To tackle this problem, use *synchronized* keyword which enables a thread to safely acquire and release the resource.