SOFTWARE ENGINEERING (CSC 403)

QUESTION:

WRITE ABOUT THE CHARACTERISTICS OF A GOOD SOFTWARE **"CORRECTNESS".**

GROUP 4

CSC/20U/4019	CSC/22D/4472
CSC/20u/19u/3611	CSC/21D/3214
CSC/20U/4000	CSC/22D/4461
CSC/20U/4104	CSC/22D/4442
CSC/20U/4074	CSC/20U/4156
CSC/20U/3986	CSC/20U/4001
CSC/20U/3980	CSC/20U/4094
CSC/22D/4458	CSC/22D/4439
CSC/20U/4097	CSC/20U/4040
CSC/22D/4462	CSC/20U/3978

CORRECTNESS:

Is a fundamental characteristic of software that refers to the extent to which the software performs its intended functions accurately, reliable, and consistently. Ensuring the correctness of the software design being prepared is very important. By correctness we mean it should accurately implement all the functionality that software product needs to exhibit.

The design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

A software design which is not correct will never help you to develop good software product. Suppose if the software designer wrongly captures the relationship between different modules or wrongly interprets the data flow between two modules then in the next phase software coders will develop the software product based on the design. And the end software product is going to have anomaly in it as it would never behave as captured in SRS document during software requirement and specification phase. This would result in re-work and thus add unnecessary cost to the software development.

Thus a software designer should be very cautious while designing the layout for the project and make sure to ensure correctness.

TYPES OF CORRECTNESS:

1. **Functional correctness:** the software performs its intended functions, and the output is accurate and consistent.
2. **Logical correctness:** the software's internal logic and algorithms are correct, ensuring that the output is valid and consistent.
3. **Semantic correctness:** the software's output is meaningful and consistent with the expected results.

FACTORS AFFECTING CORRECTNESS:

1. **Design:** flaws in software design can result in logical and functional correctness issues.
2. **Implementation:** programming errors, such as syntax errors or logical mistakes, can affect correctness.
3. **Testing:** inadequate testing can fail to detect correctness issues.

EFFECT OF INCORRECT SOFTWARE

- Data corruption
- System crashes
- Security vulnerabilities
- Financial losses

EXAMPLES OF SOFTWARE THAT FAILS DUE TO INCORRECTNESS

- **British Passport System (1999)**

The United Kingdom passport agency started a new computer system, which failed to issue passports on time to over half million citizens. Later, the agency paid millions in form of compensation, staff overtime and umbrellas for people queuing in the rain for passport.

Reason: The agency rolled out their software without proper testing and without training its employees about the new system. Also, the new law (released at the same time) required all children under 16 travelling abroad to get a password. This resulted in a huge spike in passport demand, overloading the amateur software system.

- **Mariner 1**

The Mariner 1 (1962) spacecraft headed for Venus diverted from its intended path after 293 seconds of liftoff. The mission was completed by Mariner 2 which launched 5 weeks later.

Reason: This is a combination of two failures – an antenna hardware failure and an onboard guidance system software failure. The guidance antenna performed below its specification. So, the spacecraft had to rely on its onboard guidance system, which had a bug in it.
A programmer incorrectly transcribed a formula into computer code, missing a single subscript bar, which was meant for nth smoothed value of the time derivative of a radius R. Without this smoothing function the software treated normal variation of velocity as if they were serious, causing vague corrections that sent the spacecraft off course.

- **Hartford Coliseum Collapse**

The Hartford Civic Center Coliseum collapsed on January 18, 1978, just hours after nearly 5 thousand spectators left the Coliseum. The steel-latticed roof collapsed under the weight of snow.

Reason: There were many conflicting accounts of failure, including design flows, construction and programming errors. The CAD programmer designed the coliseum incorrectly, assuming that the roof supports would only face pure compression.
Also, the computer model assumed all of the top chords were laterally braced, but in fact only interior frame met the criteria. Dead loads were underestimated by more than 20%. When one of the supports unexpectedly buckled from the snow, it sets off the chain reaction that brought down the other roof sections.

WAYS TO ENSURE CORRECTNESS IN SOFTWARE

1. **Outline Clear Requirements**
   This means making a clear requirements from the start so everyone on the team understands exactly what the software should accomplish. When requirements are too vague or keep shifting, developers may build features that don't match the real needs of the project. This can result in extra work, missed deadlines, or confusion about the final product's design.

2. **Utilize Version Control**
   Ensuring quality in software development benefits from version control, which keeps your project organized by tracking every code change. This approach clarifies who modified each file and when, providing insight into decision-making. It also reduces conflicts when multiple team members work on the same codebase.

3. **Implement Code Reviews**
   Code reviews help teams spot potential issues in the software development process before they become bigger problems down the line. When one or two teammates look over your work, they might notice inconsistencies, unused variables, or missed edge cases that you wouldn't see on your own. This extra set of eyes also encourages everyone on the

team to follow the same coding style, making the project easier to understand and maintain.
Reviewing code also creates an environment where learning and sharing best practices can happen naturally. Developers can offer tips on how to write clearer functions or suggest ways to handle corner cases better. Over time, this feedback loop helps improve the overall quality of the project and keeps the codebase more organized.

4. **Embrace Testing at Every Stage**
   Regular software testing methods help confirm that the software is working properly at every step. Automated checks can verify that a login feature remains functional whenever new code is introduced, preventing unexpected breakdowns. Meanwhile, manual testing allows you to experience the user interface in a realistic way, catching design issues or confusing workflows that automated tests might not detect.
   By integrating tests throughout the development process, you can spot potential problems early rather than letting them accumulate. This approach also makes it more comfortable to add new features, since any conflicts or errors will be noticed quickly. Whether it's unit testing, integration testing, or user acceptance testing, covering multiple layers of the software often results in a more reliable and user-friendly product.

5. **Keeping Documentation Updated**
   Updated documentation is part of software maintenance practices that helps new developers understand the project faster, reducing the chance of misunderstandings. It can also save everyone time by clearly outlining how each part of the application works, so developers don't have to guess or repeatedly ask questions.