CSE 464 – Homework 1
Bhavya Patel - ASU ID: 1225740997

Q1) a)

- Functional suitability: How well the software's features cover the required tasks, return correct results, and are appropriate for the intended use under stated conditions.
- Performance Efficiency: How fast the system responds and how much CPU, memory, and network it consumes to deliver that speed for expected workloads.
- Compatibility: How well the product coexists and exchanges information with other systems, components, and environments without adverse effects.
- Usability: How effectively, efficiently, and satisfactorily target users can learn, operate, and succeed with the interface in its real context of use.
- Reliability: How consistently the system performs its functions over time, stays available, tolerates faults, and can recover from failures.
- Security: How well data and operations are protected so only authorized entities can access or change them, with confidentiality, integrity, and authenticity enforced.
- Maintainability: How efficiently the software can be analyzed, changed, tested, and reused with low risk and effort as it evolves.
- Portability: How easily the product can be adapted, installed, or replaced across platforms and environments without excessive rework.

b) i)

Measurable factor	**ISO 25010 category**
Time behavior	Performance efficiency
Adaptability	Portability
Recoverability	Reliability
Interoperability	Compatibility
Data Integrity	Security
Learnability	Usability
useful help features	Usability
Availability	Reliability
Modifiability	Maintainability
Confidentiality	Security

ii)

- **Time behavior:** Set an SLO so p95 end-to-end checkout finishes in ≤ 30 s, with per-step limits for login, cart updates, save cart, and payment. Validate with scripted load tests that track latency, throughput, and errors, then use traces and server metrics to find and fix bottlenecks until both step and end-to-end targets pass.

- **Confidentiality:** Enforce TLS everywhere, strong password hashing, hardened sessions, and strict authorization so accounts, carts, and payment data stay private. Probe with risk-based tests and pentests for IDOR, auth bypass, and data exposure; ensure no sensitive fields appear in responses and that denied access is logged.
- **Recoverability:** Treat the 30-minute maximum downtime as the recovery time objective and design for 24/7 operations with failover. Run failover/restore drills, measure time to recover, and verify carts/orders persist or can be rebuilt within the window.

Q2

a) This video about Exploratory testing (ET) talks about staying flexible. Instead of just following a strict script, we're learning, designing, and executing tests simultaneously. This lets us chase down anything that looks a little off, unlike those pre-written tests that can make us miss something important because you're locked into a specific path. It helps you fight "inattentional blindness" by encouraging us to adapt as we go. We always look for new risks and weird behaviors as the product changes. It's great for spotting subtle bugs and inconsistent user experiences, since it encourages us to compare what we see to what we expect from a good product. With ET, we're free to push into those risky areas often ignored by scripted tests—things like heavy memory usage, OS quirks, or complicated inputs. It's about finding and giving those hidden features a real workout with challenging data and sequences. That's where we often see the strange crashes, data loss, and other disruptive issues that a simple test script would never uncover. In short, this video helped us understand that ET is the perfect partner for regular testing because it lets us test the system in a more realistic and messy way, revealing bugs that a static script would never anticipate.

b) Testing MS Word with an exploratory approach would be about jumping in and trying things out, all while learning as you go. I wouldn't use a pre-written test script; but would constantly be designing, executing, and learning at the same time. First, I'd figure out the main purpose of MS Word, which is to create and edit documents. This would help me identify the primary functions, then, I'd explore the software to find other contributing functions. I'd also be looking for areas that seem risky, like functions that handle lots of data or interact with the operating system. The entire process would be a cycle of testing, discovering something new, and then deciding what to test next based on that information.

Primary and contributing functions:

- Create/open/save documents
    - autosave
    - export to PDF
    - version history
- Text editing (insert/delete/select)
    - undo/redo
    - clipboard operations
    - navigation pane
- Text and paragraph formatting
    - Styles
    - Themes
    - Text Align
- Page layout and printing
    - Margins/orientation
    - Columns
    - Print Preview
- Find and replace
    - Advanced find options
    - whole-word/case matching
    - use of wildcards.
- Review
    - compare/merge
    - word count
    - editor suggestions
- References
    - Bibliography
    - Captions
    - Cross-references
- Tables
    - table styles
    - Sorting
    - header row repeat

- Insert media (images/shapes)
    - image wrapping
    - alt text
    - compression
- Spelling/grammar
    - Thesaurus
    - readability suggestions
    - language settings

Primary: Find & Replace

I would assess Find & Replace by crafting a document full of recurring terms, mixed casing, and words embedded within others, then confirm the tool locates the intended targets and only changes what it should across single, selection-only, and whole-document runs. I would verify that options like whole-word, case sensitivity, and wildcards behave as expected, and that formatting, links, and headings remain intact after saving, closing, and reopening the file. Furthermore, I would try stress cases such as huge files, thousands of replacements, replacements inside headers/footers and text boxes, and mid-operation interrupts (undo/redo or autosave) to see whether the feature stays responsive and consistent without corrupting content.

Contributing: Word Count

To evaluate Word Count, I would prepare documents that mix body text, tables, text boxes, footnotes, comments, and tracked changes, then compare totals for selection versus entire document to ensure the scope is honored. I would also check that numbers shown on the status bar agree with any dialogs or reports, and that counts refresh quickly after edits, accepting/rejecting revisions, or toggling markup visibility. I would additionally explore edge situations like hidden text, extremely long documents, and rapid context switches to confirm that the totals neither stall nor drift, and that results persist correctly after save and reopen.

c) Tools that support ET:

   a. Testpad: A lightweight, checklist-style test manager that fits exploratory charters, quick note-taking, and on-the-fly organization; keyboard-driven flows and flexible lists support rapid ideation and evolving coverage without over-scripting. This helps testers keep sessions focused while retaining the freedom to branch, record findings, and convert discoveries into shareable artifacts.
   b. Xray: Provides session charters and time tracking, rich evidence capture (video, screenshots, notes, annotations), and tight Jira/Xray integration to file bugs and export session reports, which directly supports ET's

reviewable-results mandate and smooths collaboration. These features let testers focus on learning and probing while automatically preserving an auditable trail of what was explored and what was found.

c. Azure DevOps Test & Feedback extension: Enables connected or standalone exploratory sessions with annotated screenshots, screen recording, user action logs, system info, page load data, and one-click work item creation for bugs, tasks, and test cases, plus session exports and end-to-end traceability. This lines up with ET's concurrent design/execution loop and "reviewable results," making it easy to capture context, evidence, and links back to requirements while staying lightweight and adaptive.

Q3 a)

1. (S ∩ P ∩ T): Specified, implemented, and tested— the spot where requirements are built and verified. Expansion of this region increases justified confidence.
2. (S ∩ P, not T): Specified and implemented but untested— a coverage gap where regressions and edge cases can slip through despite being done on paper.
3. (P ∩ T, not S): Implemented and tested but not specified— undocumented behavior that might be valuable, risky, or a defect. Tests here can legitimately check emergent or unintended behavior.
4. (S ∩ T, not P): Specified and covered by tests, but not implemented— failing or blocked features that indicate open defects or incomplete builds.
5. (S only): Specified but neither implemented nor tested— requirements debt and faults of omission waiting to happen.
6. (P only): Implemented but neither specified nor tested— undocumented features, dead code, or accidental behavior, often tied to faults of commission and high risk.
7. (T only): Tests with no corresponding specified or implementation— possibly obsolete or misaligned cases, but can also serve to assert the absence of unwanted behavior.
8. (outside all): Behaviors neither specified, implemented, nor tested— the unknowns that highlight blind spots and future discovery work.

b) No, because a solid mix of functional (specification-based) and structural (code-based) tests won't catch everything. They leave some major risks, like new emergent behaviors, weird interactions with the environment and OS, and problems for the end-user that were never part of the original design. As the product evolves, so do the risks. Relying only on a fixed, scripted approach can lead to "inattentional blindness," causing to miss issues outside our predefined inputs and paths. This is where exploratory and scenario-based testing is really better. They're all about adapting to new information and actively probing areas of potential instability, such as interoperability, error recovery, and handling complex data.

Work Cited

Bach, James. General Functionality and Stability Test Procedure for Certified for Microsoft Windows Logo: Desktop Applications Edition. Satisfice, Inc., 26 Aug. 1999.
Kaner, Cem. Black Box Software Testing: Exploratory Testing. Florida Institute of Technology, 2006.
“Install the Test & Feedback Extension.” Microsoft Learn, 7 Feb. 2025, https://learn.microsoft.com/en-us/azure/devops/test/perform-exploratory-tests?view=azure-devops
“Exploratory Testing App.” Xray, 31 Oct. 2021, https://www.getxray.app/exploratory-testing
“10 Best Exploratory Testing Tools.” Testsigma, 26 Nov. 2023, https://testsigma.com/blog/exploratory-testing-tools/
Manioudakis, Stelios. “Requirements, Code, and Tests: How Venn Diagrams Can Explain It All.” DZone, 4 Feb. 2024, https://dzone.com/articles/requirements-code-and-tests-how-venn-diagrams-can