Failure analysis is crucial in preventing and identifying issues that could lead to system failures in various fields of engineering, including computer science. In computer science, a common approach to failure analysis is through preventative testing. Preventative testing seeks to identify potential issues early in the development process to avoid system crashes or security breaches later on.One of the most widely used methods of preventative testing in the software engineering field is regression testing. Regression testing ensures that new code changes, such as bug fixes or feature additions, do not disrupt previously functioning code [1]. This method was developed to solve the problem of unintended side effects from code changes, where a new feature might break other parts of the software that were previously verified to be working. The process involves running a suite of tests on both the old and new code to ensure no regressions in functionality have occurred [1]. This type of testing is essential in software maintenance and is particularly helpful when automated to quickly identify failures in complex codebases.Another preventative method is dynamic code analysis, which involves testing software during runtime. Unlike static analysis, which inspects code without executing it, dynamic analysis identifies failures that occur only during execution, such as memory leaks, performance issues, or resource exhaustion [2]. This method provides insights into the actual behavior of the software, allowing engineers to catch problems that might not be evident from the code itself [2].In forensic failure analysis, a commonly used method is crash dump analysis. When an application fails, it generates a crash dump that captures the system’s state at the moment of failure which Microsoft showed an example of a file that writes this information called “MiniDumpWriteDump”[3]. Engineers analyze this file to determine the root cause of the crash. This method is essential for diagnosing complex system failures and helps in identifying hard-to-reproduce bugs or security vulnerabilities.

References:

[1] “Performance Regressions Policy.” *Mozilla*, www.mozilla.org/en-US/about/governance/policies/regressions/. Accessed 25 Sept. 2024.

[2] McConnell, Steven C. *Code Complete, Second Edition Ebook*, 2004, people.engr.tamu.edu/slupoli/notes/ProgrammingStudio/supplements/Code%20Complete%202nd.pdf.

[3] “Crash Dump Analysis - Win32 Apps.” *Win32 Apps | Microsoft Learn*, 9 July 2024, learn.microsoft.com/en-us/windows/win32/dxtecharts/crash-dump-analysis.