Q1

After reading the two papers provided, I believe these are the two most essential factors making software testing activities difficult:

1. It's difficult to obtain full code coverage and test all scenarios a program may go through, and software requirements may change. In order to obtain full code coverage, a tester often needs to test both valid and invalid inputs. A tester also needs to test random input to make sure a program never fails. Furthermore, if a program is set to be used by thousands of users, it is difficult to simulate such high traffic through testing. Often times, code may also be written without testing in mind, making it very difficult to test. All of these factors make it very time consuming to do a decent/complete job of testing. Sometimes, the time spent on creating tests can be more than the time spent on the development of the actual software. Furthermore, when software requirements change, tests may break during development. It may be difficult and costly to fix test cases. This is a technical factor.

2. Developers may not inherently be that interested in the testing part of software development. For many developers, writing tests is not an enjoyable part of development, and may even be seen as a chore. From personal experience, successfully writing tests for a program is nowhere near as enjoyable or rewarding as the feeling of creating the program being tested. Also, software testing is not something that is as celebrated as software development. While most developers may agree that software testing is important, they may not be very motivated to do software testing. As a result, it's possible developers may slack when it comes to testing (or even forgo it altogether), perhaps thinking that the piece of code they wrote already works and that writing a test is just extra work. When testing isn't high in the priority list for a developer, they may also create code that is not very testable in nature. This is a non-technical factor.

Q2

T-Mobile wireless services unavailable for 12 hours	
Failure description	On June 15, 2020, T-Mobile experienced an outage that lasted over 12 hours. This disrupted calling and texting services worldwide (including at least 23,621 failed 911 calls). At least 250 million calls were disrupted by the outage. The outage started from an equipment failure that was exacerbated by a network routing misconfiguration that occurred when a new router was introduced into T-Mobile's network.

Nature of software failure	Routers in T-Mobile's network utilize a routing protocol called Open Shortest Path First where each set of lines (called links) that connects T-Mobile's routers is assigned a weight. The network then decides where to send LTE traffic by selecting the route with the lowest total weight. T-Mobile was installing routers in the southeast region of its network. T-Mobile installed the first of the routers to be installed and planned to install the second once the first was active and handling traffic. T-Mobile planned to install this second router as a passive router such that it would only receive traffic if another router or link between routers failed. However, T-Mobile misconfigured the links of another router that was already in the network but was not designed to process call traffic. This misconfiguration meant that in the event of a router or link failure, this router would receive a large percentage of call traffic (which it could not handle). T-Mobile did not have a fail-safe process to prevent or detect this misconfiguration. Later in the day, a fiber transport link failed. Under regular circumstances where router weights were properly configured, this would have been mitigated. However, due to the misconfiguration of weights mentioned earlier, the router that was not meant to handle high call traffic was overloaded. The Atlanta network became isolated from the rest of the network, causing all LTE users in the area to lose connectivity. A software error caused further issues by preventing devices from re-registering with the IP Multimedia Subsystem over Wi-Fi. This error meant that instead of routing device registration attempts to a different node, attempts were routed to unavailable nodes. This software error had existed for months in T-Mobile's networks. However, it did not cause issues until the outage occurred. Eventually, the Atlanta outage spread nationwide as the software error continued to send registration attempts to nodes that were already severely congested.

Any testing efforts regarding the failure?	T-Mobile created separate communication channels to enable management of affected routers even during an outage. T-Mobile also introduced additional dedicated 911 nodes for better resiliency. The error messages for down nodes was also improved for better troubleshooting. New software updates were pushed to improve node robustness and multiple systems were audited for potential enhancements.
Any follow up action taken? Any plan to alleviate further problems?	T-Mobile corrected the Open Shortest Path First weights. T-Mobile also corrected the software issue discussed earlier. The FCC also recommended that service providers should validate node software and router integration in a test environment to discover software flaws and routing misconfigurations.
URL	(1) `https://www.reuters.com/article/us-t-mobile-us-outage/june-t-mobile-u-s-network-outage-disrupted-more-than-250-million-calls-fcc-idUSKBN2772B3` (2) `https://docs.fcc.gov/public/attachments/DOC-367699A1.pdf` (2) `https://arstechnica.com/tech-policy/2020/10/fcc-not-punishing-t-mobile-for-outage-that-ajit-pai-called-unacceptable/`

Q3

Software Quality	**Meaning**
Functionality	Does the software fulfill its purpose (can drive without human input, driving safely on any road/route, taking the correct and optimal path, driving comfortably, allowing human override)?
Reliability	Does the software reliably avoid obstacles regardless of situation (such as bad weather)? Does the software have the capability to avoid collisions? Does the software perform under challenging scenarios (parking lots, heavy traffic)?
Portability	Does the software work for a variety of cars? Can the software work alongside other software? Can the software run on different architectures (x86, ARM, etc.)?

Efficiency	Does the software utilize hardware in a reasonable fashion (does it use the right amount of CPU, GPU, etc.)? Is the software fast enough to be responsive and safe? Is the software using the right algorithms to maximize compute efficiency?
Maintainability	Is the software documented? How easy is it for a new programmer to learn the codebase? Is there consistent styling/programming paradigm (OOP, functional, etc.)? Are there unit tests and is the code easily testable? Is the code structured well to allow for easy addition of new features?

Risk	Technical Risk	Business Risk
Car does not drive safely, does not obey traffic laws, or is otherwise unsafe	2	1
Car often requires human input	4	2
Car does not allow human override	5	2
Car does not drive comfortably	5	2
Car does not take optimal paths	3	2
Car does not utilize resources efficiently, program freezes, or program responds slowly	1	1
Software code is messy, inconsistent, and is extremely difficult to understand and build upon	1	4
Software is not documented correctly	2	3
Software is not easily testable or does not have unit tests	1	5
Software is limited to a single or a few cars only	3	2
Software only runs on a single architecture (ex. x86, ARM, etc.)	3	4