Project 3 Part I - Risk Based Testing

1. **Priority Testing**

Test legacy code

Risk Exposure= RE = P(failure) * consequences = HIGH/MEDIUM ("code tends to crash much of the time", "software compatibility issues with rest of code")* CATASTROPHIC("an error will bring down app for multiple days, rendering site useless", heavy impact on usability) = HIGH

Test outsourced code

RE = P(failure) * consequences = LOW ("well-documented code", "good job with their development", "low likelihood for the page to stop working", "well tested") * MARGINAL("application always available", "minimal damage if section was not working", "usability not affected") = LOW

Test APIs

RE = P(failure) * consequences = HIGH("likelihood of API going down very high") * CRITICAL("if API goes down application still running, but performance would be slower, users wouldn't get updated information") = MEDIUM

Test new code

RE = P(failure) * consequences = LOW("experienced developers", "well-documented", "easy to test", "low likelihood of having significant number of errors") * CATASTROPHIC("heavy impact on availability and usability", "other portions of application also can't be used") = MEDIUM

Priority Testing

1. Test legacy code
2. Test APIS
3. Test new code
4. Test outsourced code

   Explanation : Given the calculated risk exposure, we should start testing the legacy code first to mitigate the highest risk, followed by testing the APIS, then testing the new code and the outsourced code. The last two testing tasks should be easier to complete because the code is well-documented/well-developed, tests have been written/are easy to write and the code is easy to test.

Note : Even if "APIS" and "New code" both have a medium risk exposure, I would still test first the APIS over the NEW Code given that the likelihood of failure in the APIS case is > likelihood of failure for the new code.

2. Intensity Ranking

Test legacy code (Risk exposure HIGH) : maximal testing
Test outsourced code (Risk exposure LOW) : minimal testing
Test APIs (Risk Exposure MEDIUM) : average testing
Test new code (Risk Exposure MEDIUM) : average testing

3. Contingency Plan

Failure in	*Contingency plan*
Legacy code	Rollback to a previous stable release to get make app available & working again. Deploy measures from preventing this happen again : Plan & start testing legacy code (unit testing in particular), testing is a great way to find out what the app does. Plan & start refactoring the legacy code bits & pieces, maybe start with those parts in the legacy code that "communicate" with the rest of the app to improve compatibility.
Outsourced code	Run all tests again, see if there were any failures. Try to isolate and fix the error/errors using the documentation & tests. Try to write new test cases to avoid the bug/errors in the future.
APIs	Run existing tests to check functionality of the APIs (like correct validation of requests etc). Perform non-functional testing : performance, stress & volume testing and try to determine resource bottlenecks. Eliminate bottlenecks.
New code	Run existing test suites, see where they fail. Find, isolate & fix bugs (using documentation & tests). Write new test cases from found bugs/errors.