CODING STANDARDS:

* **Choose industry-specific coding standards**

Coding best practices and standards vary depending on the industry a specific product is being built for. The standards required for coding software for luxury automobiles will differ from those for coding software for gaming.

For example, MISRA C and C++ were written for the automotive industry and are considered the de-facto standards for building applications that emphasize safety. Currently, they are regarded as the absolute best practices for writing code in the industry.

Adhering to industry-specific standards makes it easier to write accurate code that matches product expectations. It becomes easier to write code that will satisfy the end-users and meet business requirements.

* **Focus on code readability**

Readable code is easy to follow, optimizes space and time. Here are a few ways to achieve that:

1. Write as few lines as possible.
2. Use appropriate naming conventions.
3. Segment blocks of code in the same section into paragraphs.
4. Use indentation to marks the beginning and end of control structures. Clearly specify the code between them.
5. Don’t use lengthy functions. Ideally, a single function should carry out a single task.
6. Use the DRY (Don’t Repeat Yourself) principle. Automate repetitive tasks whenever necessary. The same piece of code should not be repeated in the script.
7. Avoid Deep Nesting. Too many nesting levels make code harder to read and follow.
8. Capitalize SQL special words and function names to distinguish them from table and column names.
9. Avoid long lines. It is easier for humans to read blocks of lines that are horizontally short and vertically long.

* **Standardize headers for different modules**

It is easier to understand and maintain code when the headers of different modules align with a singular format. For example, each header should contain:

1. Module Name
2. Date of creation
3. Name of creator of module
4. History of modification
5. Summary of what the module does
6. Functions in that module
7. Variables accessed by the module

* **Don’t use a single identifier for multiple purposes**

Ascribe a name to each variable that clearly describes its purpose. Naturally, a single variable can’t be assigned multiple values or used for numerous functions. This would confuse everyone reading the code and would make future enhancements more difficult to implement. Always assign unique variable names.

* **Turn daily backups into an instinct**

Multiple events can trigger data loss – system crash, dead battery, software glitch, hardware damage, etc. To prevent this, save code daily, and after every modification, no matter how minuscule it may be. Back up the workflow on TFS, SVN, or any other version control mechanism.

* **Leave comments and prioritize documentation**

Don’t assume that just because everyone else viewing the code is a developer, they will instinctively understand it without clarification. Devs are human, and it is a lot easier for them to read comments describing code function rather than scanning the code and making speculations.

Take an extra minute to write a comment describing the code function at various points in the script. Ensure that the comments guide any readers through the algorithm and logic implemented. Of course, this is only required when the code’s purpose is not apparent. Don’t bother leaving comments on self-explanatory code.

* **Try to formalize Exception Handling**

‘Exception’ refers to problems, issues, or uncommon events that occur when code is run and disrupt the normal flow of execution. This either pauses or terminates program execution, which is a scenario that must be avoided.

However, when they do occur, use the following techniques to minimize damage to overall execution in terms of both time and dev effort:

1. Keep the code in a try-catch block.
2. Ensure that auto recovery has been activated and can be used.
3. Consider that it might be an issue of software/network slowness. Wait a few seconds for the required elements to show up.
4. Use real-time log analysis.
5. **When choosing standards, think Closed vs. Open**

Consider [CERT](https://www.perforce.com/blog/kw/what-is-cert) vs. [MISRA](https://www.perforce.com/resources/qac/misra-c-cpp). CERT emphasizes community cooperation and participation. It offers a coding standard that is freely available as a web-based wiki. With CERT, users can comment on specific guidelines – comments taken into account when the standards are reviewed and updated.

On the other hand, MISRA is a set of C and C++ coding standards developed and maintained by the Motor Industry Software Reliability Association (MISRA). It is primarily considered the de-facto coding standard for embedded industries.

MISRA was created and is updated by working groups according to predetermined blueprints. While secure and reliable, it is not available for free though it does admit some community feedback when implementing updates.

Naturally, CERT is easier to work with. But open standards change quickly, which can make them hard to keep up with. However, closed standards like MISRA are better for safety-critical industries because they enforce uniformity across teams, organizations, and vendors. In other words, they offer a reliable reference that demands adherence to a set of mandatory requirements. With products like those created by the automotive industry, standards need to be set in stone because real lives are at stake.

Despite adhering to the coding best practices detailed above, bear in mind that all code needs to be extensively tested on real browsers and devices. Instead of dealing with [emulators and simulator](https://www.browserstack.com/emulators-simulators)s’ many inadequacies, testers are better off using a real device cloud that offers real devices, browsers, and operating systems on-demand for instant testing.

By running tests on a [real device cloud](https://www.browserstack.com/real-device-cloud), performance tests can be conducted to ensure that they are getting accurate results every time. Comprehensive and error-free testing ensures that no major bugs pass undetected into production, thus enabling software to offer the highest possible levels of user experience.