School of Engineering and Applied Science,
Ahmedabad University

CSE300 Software Engineering
Coding Standards Report

Guided By: Prof Khusru Doctor
TA Muskan Matwani
TA Anupama Nair

TRAVOYAGER

We are here to get you there

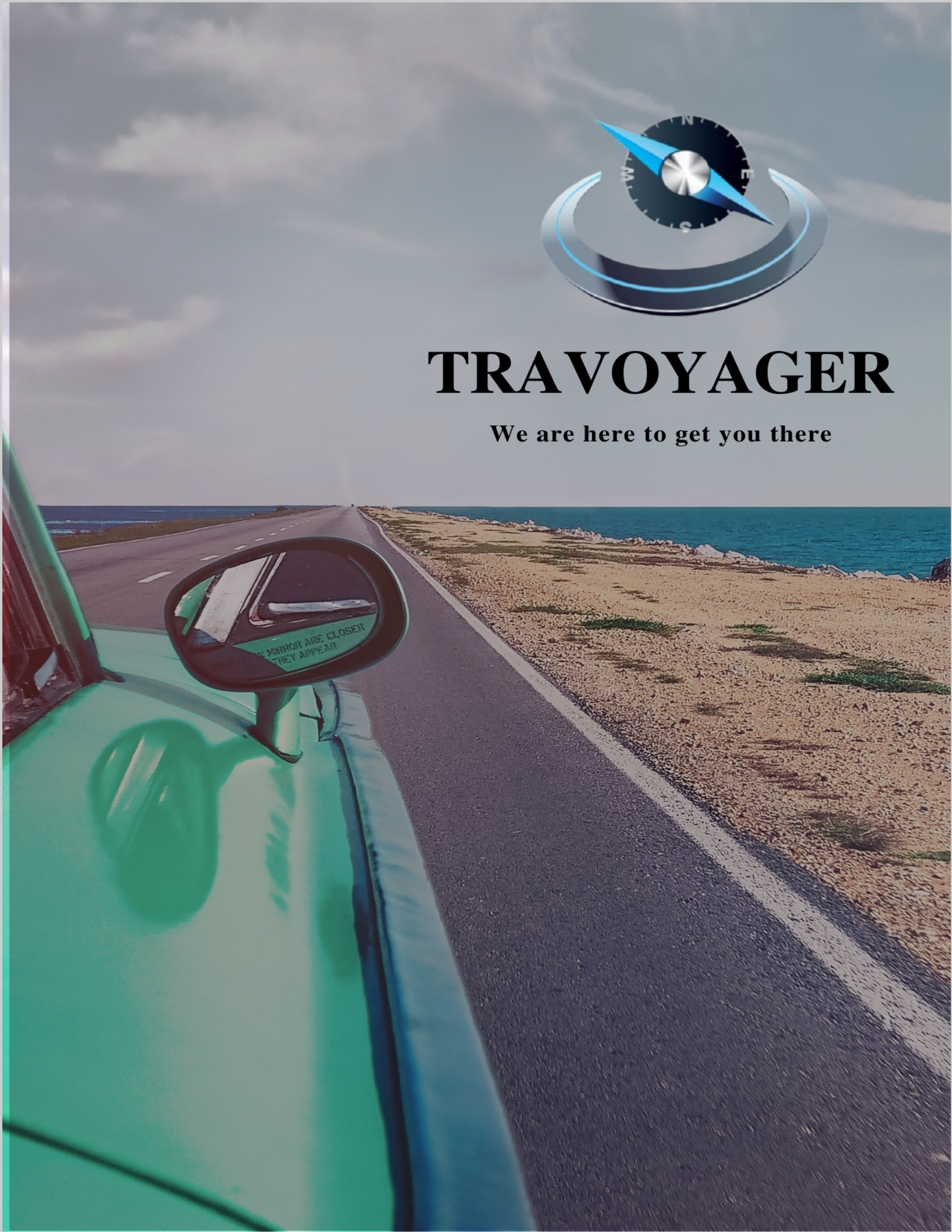

Table of Contents

1. What are coding standards?

Coding standards are collections of coding rules, guidelines, and best practices. Coding standards will help you to write cleaner code.The main goal of the coding phase is to code from the design document prepared after the design phase through a high-level language and then to unit test this code.

Good software development organizations want their programmers to maintain some well-defined and standard style of coding is called coding standards. Development organizations usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop. It is very important for the programmers to maintain the coding standards otherwise the code will be rejected during code review.

2. Why are they required?

Coding standards help in the development of software programs that are less complex and thereby reduce the errors. If the coding standards are followed, the code is consistent and can be easily maintained. This is because anyone can understand it and can modify it at any point in time. It can be interpreted as a set of procedures for a particular language of programming that defines the style of programming, procedures, methods for different aspects of the software written in that language. They are a very important Software Development attribute.

Purpose of Having Coding Standards:

1. A coding standard gives a uniform appearance to the codes written by different engineers.
2. It improves readability, and maintainability of the code and it reduces complexity also.
3. It helps in code reuse and helps to detect error easily.

4. It promotes sound programming practices and increases efficiency of the programmers.

2.1. Advantages

1. Reduced complexity :
   Coding guidelines are maintained properly, then the software code increases readability and understandability thus it reduces the complexity of the code.
2. Enhanced Productivity:
   It is also shown that software developers invest most of their time addressing the issues that should have been avoided. Implementing the coding standards will help the team identify the problems early, or even fully eliminate them. This will increase efficiency throughout the software process.
3. Risk of project failure is reduced:
   It occurs several times when IT projects struggle when designing software due to problems. Implementation of the coding standards reduces many problems and the risk of failure of projects.
4. Cost efficient:
   A simple code gives developers the possibility to reuse the code whenever appropriate. It, along with the efforts put into growth, will dramatically reduce costs.

2.2. Disadvantages

1. Standards force people to change their methods which are perfectly alright as they are.
2. Standards reduce productivity by forcing unnecessary actions.
3. Standards do not prevent bugs.
4. Contradictions, bugs, and logic flaws are the principal causes of widely exploited software vulnerabilities. Most of these problems arise due to programming errors that result from poor coding practices.
5. Poor coding standards affect web efficiency overall.
   a. Reusability of the user interface
   b. System response code
   c. Flow problems

3. Python(Django) Standards

We will follow the pep8 and Google python style guidelines as standard.

1. Naming Convention
   a. Use meaningful names
   b. Function and variable names in snake_case
   c. Classname in PascalCase
   d. Constants snake_case capitalized

2. Indentation/Space

   a. Use 4 spaces for indentation(Python 3 disallows mixing the use of tabs and spaces for indentation)
   b. Separate top-level function and classes with two blank lines
   c. Separate method definition inside the class by one line
   d. The maximum length of the line should be less than 80 characters
   e. There should be no trailing white spaces

3. Imports
   a. Import from Python standard library(1st)
   b. Import from core Django(2nd)
   c. Import from 3rd party vendor(3rd)
   d. Import from Django Apps(4th)(Current Project)
   e. Avoid import *

4. Migrations
   a. Do not modify the files created by make migration command(do not add custom SQL command)
   b. Place custom SQL command if needed in a separate file and do not mix it with the auto-generated files from make migration command
   c. Forward and backward migration work only on auto-generated files by make migration command
   d. Not all migration can be reversed
   e. Add — database=<dbConfigName> always in your migration

5. Response Status
   A response message with status codes
   Example,
```
{
        'status':'success|error',
        'data':{
                'result':{} || [] , ''
        }, #one level
        'meta':{} #any meta information that you want to pass
}
```

6. Template style
   In the Django template code, put one (and only one) space between the curly brackets and the tag contents.
   Example:
   {{ foo }}

7. View style
   In Django's views, the first parameter in a view function should be called request.
   Example:

def my_view(request, foo):
…

8. Model style
   Field names should be all lowercase, using underscores instead of camelCase.
   Example:
   class Person(models.Model):
   first_name = models.CharField(max_length=20)
   last_name = models.CharField(max_length=40)

4. HTML Standards

1. Coding Style Guidelines
   Consistency is absolutely a prerequisite for maximizing maintainability and reusability. These general guidelines for coding style can form the basis of a set of standards that will help ensure that all developers in a project—or, better, in all projects across an organization—write code consistently.
   a. Use well-formed HTML.
   b. Pick good names and ID values.
   c. Indent consistently.
   d. Limit line length.
   e. Standardize character case.
   f. Use comments judiciously.

2. Using UTF-8 (No Byte Order Mark)
   Making sure to use UTF-8 as character encoding, without a byte order mark.

3. Using Well-formed HTML
   a. Lowercase names—To be well-formed, element and attribute names must be in all lower case. In versions through 4.01, HTML is not case-sensitive. However, XML is case-sensitive, and it follows that the XHTML 1.0 recommendation is also case-sensitive. So, to ensure that the code keeps working and to maximize reusability, this must be planned for.
   b. Closing tags—All nonempty elements must have corresponding closing tags. Empty elements—those previously signified with a single tag, such as **<hr>** and **<br>**—must be followed immediately by a corresponding closing tag, or the tag must end with "/". For example, **<hr></hr>** and **<hr/>** are both examples of well-formed code.
   c. Nested elements—All nested attributes must be properly nested—for example:
      <center><b>Some text</b></center>
      Note that the <b> tag and its corresponding closing tag, </b>, are both nested inside the <center> and </center> tags.
      -If elements overlap, then they are not properly nested, as illustrated in the following code:

<center><b>Some text</center></b>
While many browsers have accepted overlapping elements and given the expected results, they have always been, strictly speaking, illegal in HTML, and future versions of browsers might not support them.

   d. Attribute values—Attribute values, even numeric attributes should be quoted—for example:
   <input name="txtName" type="text" size="1">

4. Code validation: Another step toward improving HTML code is to validate it against a formal published grammar and to declare this validation at the beginning of the HTML document. For example, the following line declares validation against the public HTML 3.2 Final grammar:
   <!doctype html public "-//W3C//DTD HTML 3.2 Final//EN">

5. Indent Consistently
   Use indentation consistently to enhance the readability of the code. When elements carry over more than one line of code, indent the contents of elements between the start tag and the end tag.This will make it easy to see where the element begins and ends. Also, use indentation to align code at attribute names.

6. Semantics
   a. Use HTML according to its purpose.
   b. Use elements (sometimes incorrectly called “tags”) for what they have been created for. For example, use heading elements for headings, p elements for paragraphs, elements for anchors, etc.
   c. Using HTML according to its purpose is important for accessibility, reuse, and code efficiency reasons.
   <!-- Not recommended →
   <div onclick="goToRecommendations();">All recommendations</div>
   <!-- Recommended -->
   <a href="recommendations/">All recommendations</a>

7. Multimedia Fallback
   Provide alternative content for multimedia.
   For multimedia, such as images, videos, animated objects via canvas, make sure to offer alternative access. For images that means the use of meaningful alternative text (alt) and for video and audio transcripts and captions, if available. Providing alternative contents is important for accessibility reasons: A blind user has few cues to tell what an image is about without @alt, and other users may have no way of understanding what video or audio contents are about either.
   <!-- Not recommended -->
   <img src="spreadsheet.png">
   <!-- Recommended -->
   <img src="spreadsheet.png" alt="Spreadsheet screenshot.">

8. Separation of Concerns

Separate structure from the presentation from behavior. Strictly keep structure (markup), presentation (styling), and behavior (scripting) apart, and try to keep the interaction between the three to an absolute minimum. Separating structure from the presentation from behavior is important for maintenance reasons. It is always more expensive to change HTML documents and templates than it is to update style sheets and scripts.

When Quoting Attribute Values, Use Double Quotation Marks
Use double (""), not single quotation marks (''), around attribute values.
Correct:
<a class="action promo">Sign in</a>

9. Spacing
   A comma followed by an argument should be further followed by blank space. A keyword followed by braces should be separated by a space.
   Example:
   p.intro {font-family: Arial ; font-size: 15em;}
10. Comments
   Inline comments should be used in order to explain the function of any particular code snippet. This improves the readability subsequently.
   One-line comment
   <!-- This is a comment -->
   Multi-line comment
   <!--
   This is a long comment example. This is a long comment example.
   This is a long comment example. This is a long comment example.
   -->

5. CSS Standards

1. Formatting
   All CSS documents must use two spaces for indentation and files should have no trailing whitespace. Other formatting rules can be given as follows:
   a. Use soft-tabs with a two-space indent.
   b. Use double-quotes.
   c. Use shorthand notation where possible.
   d. Put spaces after: in property declarations.
   e. Put spaces before { in rule declarations.
   f. Use hex color codes #000 unless using rgba().
   g. Always provide fallback properties for older browsers.
   h. Use a one-line property declaration.
   i. Always follow a rule with one line of whitespace.
   j. Always quote url() and @import() contents.
   k. Do not indent blocks.

2. Naming
   All ids, classes, and attributes must be lowercase with hyphens used for separation.

3. Comments
   Comments should be used liberally to explain anything that may be unclear at first glance, especially IE workarounds or hacks.

4. Modularity and specificity
   Try to keep all selectors loosely grouped into modules where possible and avoid having too many selectors in one declaration to make them easy to override.

5. Property Ordering
   Above all else, choose something that is meaningful to you and semantic in some way. Random ordering is chaos, not poetry.Here, properties are grouped by meaning and ordered specifically within those groups. The properties within groups are also strategically ordered to create transitions between sections, such as background directly before color. The baseline for ordering is:
   a. Display
   b. Positioning
   c. Box model
   d. Colors and Typography
   e. Other

6. Vendor Prefixes
   We use Autoprefixer as a pre-commit tool to easily manage necessary browser prefixes, thus making the majority of this section moot. For those interested in following that output without using Grunt, vendor prefixes should go the longest (-WebKit-) to the shortest (unprefixed). All other spacing remains as per the rest of the standards.

6. Javascript Standards

1. Coding conventions are programmable design guidelines. Usually, they cover:
   a. Variables and functions naming and declaration laws.
   b. Rules for the use of white space, indentation
   c. The methods and principles of programming.

2. Secure quality of coding conventions:
   a. Improves code readability.
   b. Making code maintenance easier.

3. Variables Name:

Before they are used all variables must be declared with var. Mostly, we declare the variable at the start of the function. The names all begin with a letter.

4. Spaces Around Operators:
   Place spaces around operators such as (= +-* /) and after commas.
   Example:
   ```
   var a = b + c;
   var v = ["Abc", "XYz"];
   ```

5. Coding Indentation:
   Often use 2 spaces to indent blocks of code:
   Function:
   ```
   function Cel(f) {
           return (5 / 9) * (f - 32);
   }
   ```

6. Statement Rules:
   Always end a simple statement with a semicolon
   Example:
   ```
   var v = ["Abc", "XYz"];
   var p = {
           firstName: "kenvy",
           age: 25,
           eye: "grey"
   };
   ```

7. General rules for complex statements (compound):
   a. Place the opening bracket at the end of the first paragraph.
   b. Using one room before the bracket launch.
   c. Place the closing bracket on a new line, without spaces leading.
   d. Should not end a complex statement with a semicolon.

8. Functions:
   ```
   function Cel(f) {
           return (5 / 9) * (f - 32);
   }
   ```
9. Loops:
   ```
   for (i = 0; i < 10; i++) {
           sum += i;
   }
   ```
10. Conditionals:
   ```
   if (t < 15) {
           message = "Hello";
   } else {
   ```

```
        message = "Byee";
}
```

11. General object definition rules:
    a. Place the opening bracket on the same line as the name of an object.
    b. Use colon plus one space between its value and each property.
    c. Using quotations around string values, not numerical values.
    d. Do not apply a comma after the last pair of property-values.
    e. Place the closing bracket on a new line, without spaces leading.
    f. The definition of an object always ends with a semicolon.
    ```
    var p = {firstName:"Anvi", lastName:"Deliwala", age:25, eye:"grey"};
    ```
    LINE LENGTH < 80
    If a sentence in JavaScript doesn't fit on one line, the best place to break it is after an operator or
    a comma.
    Example:
    ```
    document.getElementById("test").innerHTML =
    "Hello World!";
    ```

12. Comments:
    Using commentaries on-site, not block comments. The comments should begin at the left-hand margin.
    Use '/' in comment startup.
    ```
    temp = 0; // Set temp to zero.
    ```

13. Naming Conventions:
    Please use all of the code with the same naming convention.
    For example:
    a. Variable and function names are written in UPPERCASE
    b. As camelCase Global variables
    c. Constants (like PI) written in UPPERCASE