

Systems and methods for comparing software code

Technical Field

[1] Embodiments generally relate to systems and methods for comparing software code, and in some embodiments, for outputting mismatches which describe the differences in compared software code.

Background

[2] Legacy code, or outdated code, may be poorly-factored and have poor test coverage making it hard to maintain. It can also be difficult to deliver new features to outdated code. However, modernising outdated code can present a number of unique issues. When changing from a monolithic frontend to a modernised micro frontend, it is desirable to have the frontend mark-up code generated by the modernised micro frontend be the same as the code previously generated by the monolithic frontend. For example, in circumstances where a monolithic endpoint is being replaced with a modernised microservice endpoint, or where technologies used inside a monolithic codebase are being upgraded (including, for example, new ways of generating frontend mark-up), it is necessary to ensure that the new output is identical to the old output. Additionally, it can be problematic to replace entire functionalities of old code with modernised code when the old code is untested, that is, where there is no “original” test or output to which the new output may be compared. As such, it can be difficult to ensure that the new output is identical to the old output where there is low test coverage of the old output. Testing and comparing legacy code and modernised code can be time consuming and laborious.

[3] Testing and comparing legacy code and modernised code can also create problems with data leakage. Some existing systems compare pieces of software code by throwing errors and logging raw results when a mismatch occurs. However, this risks exposing confidential data which may be in the form of values published in the raw

results. In some cases, the entire payload of the software code is logged, which is both difficult to interpret, and unfiltered which further risks exposing confidential data.

[4] Any discussion of documents, acts, materials, devices, articles or the like which has been included in the present specification is not to be taken as an admission that any or all of these matters form part of the prior art base or were common general knowledge in the field relevant to the present disclosure as it existed before the priority date of each claim of this application.

Summary

[5] Some embodiments relate to a computer-implemented method for comparing a first code and a second code, comprising: generating a first payload from the first code; generating a second payload from the second code; performing a comparison of the first payload with the second payload by iterating over at least one pair of nodes, the at least one pair of nodes including a first node from the first payload and a corresponding second node from the second payload; identifying at least one differing element of the at least one pair of nodes in which a difference occurs; and in response to identifying the at least one differing element, generating a mismatch object for the pair of nodes, the mismatch object defining at least one characteristic of the at least one differing element of the at least one pair of nodes, wherein the at least one characteristic includes at least a position of the differing element in the first node and the second node.

[6] In some embodiments, the method may include aggregating the mismatch object to a data set.

[7] In some embodiments, the method may include returning the first payload and/or the second payload. The method may include determining whether to return the first and/or second payload based on mismatch objects in the data set. An error may be thrown if a mismatch object is generated for a pair of nodes.

[8] In some embodiments, the first code may be parsed through a control function. The second code may be parsed through a candidate function. The first code and the second code may be idempotent.

[9] Performing a comparison of the first payload and the second payload may include applying a comparison function over each pair of nodes. The comparison function may include an analyser. The comparison function may include at least a HTML analyser function configured to compare HTML fragments. In some embodiments, the HTML compare function may be configured to normalise HTML entities.

[10] The comparison function may include a text analyser function configured to compare text. In some embodiments, the text analyser function may be configured to apply a character transform function to process whitespace.

[11] In some embodiments, the at least one characteristic may include a value type of the differing element in the first node and the second node. The at least one characteristic may include an xpath of the differing element in the first node and the second node. The at least one characteristic may include traits of the differing element. The at least one characteristic may include information about a character class of the differing element. The at least one characteristic may include information about a Unicode category of the differing element.

[12] In some embodiments, the at least one characteristic may describe the nature of the differing element in the first node and the second node. The value of the differing element may not form part of the at least one characteristic.

[13] The mismatch object may define a log message. The mismatch object may include a message describing the at least one characteristic of the at least one differing element for each of the first node and the second node. The mismatch object may

include an output of the differing element in the first payload and an output of the differing element in the second payload.

[14] In some embodiments, the data set may be a log data set. The method may include outputting the log data set as an error log.

[15] The differing element may include at least one of: elements with different tags, elements with different values, and elements in a different order. The method may include converting an HTML fragment of the first code to a string before generating a first payload. The method may include applying a setup function to configure comparison of the first payload against the second payload. The setup function may be configured to instantiate at least one of an error handler and a publisher.

[16] Some embodiments relate to a non-transient computer-readable storage medium storing instructions which, when executed by a processing device, cause the processing device to perform the method of any one of the methods described herein.

[17] Some embodiments relate to a computing device comprising: the non-transient computer-readable storage medium described herein; and a processor configured to execute instructions stored in the non-transient computer-readable storage medium.

[18] Throughout this specification the word "comprise", or variations such as "comprises" or "comprising", will be understood to imply the inclusion of a stated element, integer or step, or group of elements, integers or steps, but not the exclusion of any other element, integer or step, or group of elements, integers or steps.

Brief Description of Drawings

[19] Various ones of the appended drawings merely illustrate example embodiments of the present disclosure and cannot be considered as limiting its scope.

[20] Figure 1 is a process flow diagram of a method for comparing a first code and a second code, according to some embodiments;

[21] Figure 2 is a schematic flow diagram showing the comparison of a first code and a second code, according to some embodiments;

[22] Figure 3A is a schematic diagram showing a comparison of a pair of nodes from a first payload and a second payload, according to some embodiments;

[23] Figure 3B is a schematic diagram showing the mapping of elements in groups, according to some embodiments;

[24] Figure 4 is a block diagram of an example system architecture which performs a method of comparing the first code and the second code, according to some embodiments;

[25] Figure 5A is a diagram showing a visual representation of a first example identification of a differing element and the characteristics which are extracted to generate a mismatch object, according to some embodiments;

[26] Figure 5B is a diagram showing a visual representation of a second example identification of a differing element and the characteristics which are extracted to generate a mismatch object, according to some embodiments;

[27] Figure 6 is a flow diagram which shows a comparison of at least one pair of nodes in a first payload and a second payload, according to some embodiments; and

[28] Figure 7 is a flow diagram which shows the generation of the mismatch object after the comparison of Figure 6.

Description of Embodiments

[29] Embodiments generally relate to methods, systems and computer-readable media for comparing first code and second code, such as pieces of executable software code, to determine differences between the first code and second code and log the mismatches in a manner which characterises the differences. Software code may require comparison for various reasons, for example, during updating or modernisation of code. A comparison enables the differences between legacy code and modernised code to be readily identified. Some embodiments relate to comparing first code and second code, logging the identified differences without revealing the values of the differences, and/or returning the first code or second code.

[30] The systems and methods described herein are configured to compare a first code to a second code. Code may be referred to, and should be regarded as encompassing, source code, computer code, and/or software code. Code may encompass an entire computer program, a part of a program or object, and/or a piece of software code, for example, one or more functions, a sub-program, sub-routine and/or structured object. References to code or software code throughout the specification may refer to source code and may reference a text listing of commands or instructions written using a programming language. In some cases, code may be used to refer to source code to be compiled or assembled into an executable computer program, or after it has been processed by a compiler and made ready to run on a computer. In some embodiments, code may also be used to refer to text written for mark-up or styling languages such as, but not limited to, HyperText Markup Language (HTML) and Cascading Style Sheets (CSS). As an example, code may include text written using a programming languages including Python, Java, Ruby, C, C++, C#, Perl.

[31] A first code 200 and a second code 202 may be written in the same programming language, or they may be written in different programming languages. In some embodiments, the first code and the second code may be written in the same version of code or in different versions of code. For example, where systems and

methods are used to compare a first code 200 and a second code 202, the first code 200 may include software code written in Java Script ES5, and the second code 202 may include software code written in Java Script ES13, and the comparison may identify syntactical differences between the two versions of the code languages for the purpose of code modernisation.

[32] In some embodiments, the first software code 200 and the second software code 202 may include HTML, or HTML fragments. In some embodiments, the first software code and the second software code may be formed from an XML document or file. The first software code and the second software code may be obtained from a code path. In some embodiments, the code path may be an XSLT code path. The XSLT code path may be converted or transformed into HTML to define the first and/or second code. In some embodiments, the XSLT code path is used to return transformed HTML, and then the resulting HTML code is converted into a string.

[33] The first code 200 may define a control code, and the second code 202 may define a candidate code. The control code and the candidate code may be directly compared to identify differences between the control code and the candidate code. The differences between the first and second code may be syntactical differences in nodes or elements of the code which do not affect the outcome of executing the code, but which are not consistent between the two. The method of comparison can be used to compare legacy code to modernised code. Comparison of the first code 200 and second code 202 may be implemented in sections or fragments, for example by comparing only portions of an entire software code, or it may be performed over the whole first code and second code. Multiple portions of an entire first code 200 and/or second code 202 may be compared at the same time, or in the same comparison iteration of the method. The portions may be preselected for comparison. In some embodiments, the comparison method may be used for checking updates made to code, or for version control.

[34] In some embodiments, the first code or the control code may be an older version of a code, a legacy version of a code, a piece of code in a particular programming language, or a piece of code which is currently in use. In some embodiments, the second code or the candidate code may be a newer version of the first code or the control code, an updated version of a code, a piece of code in a particular programming language, or a piece of code which is intended to be deployed or implemented for use. The second code 202 may represent code which is intended to replace the first code 200.

[35] **Figure 1** is a process flow diagram of a method 100 for comparing a first code and a second code. To compare a first code 200 to a second code 202, payloads are generated from the first and second codes, and the payloads are used for performing the comparison. The method 100 for comparing a first code 200 and a second code 202 includes, in 110, generating a first payload from the first code. A first payload 204 may be generated from the first code 200 by inputting the first code 200 through a first function. The first function may be a control function 208, the control function being configured to return the first payload as a control payload. In some embodiments, the first code 200 is input into a control function 208, which defines the first code 200 as a control code and generates a control payload.

[36] In 120, a second payload 206 is generated from the second code 202. The second payload 206 may be generated by inputting the second code 202 through a second function. The second function may be a candidate function 210, the candidate function being configured to return the second payload as a candidate payload. In some embodiments, the second code 202 is input into a candidate function 210, which defines the second code 202 as a candidate code and generates a candidate payload.

[37] 110 and 120 may be performed in sequence or in parallel. In some embodiments, 120 may be performed before 110. By using the control function 208 with the first code 200 and the candidate function 210 with the second code 202, first and second payloads 204, 206 are respectively produced from the first and second

codes 200, 202. The first and/or second payloads 204, 206 may include the output of the first and/or second codes respectively. In some embodiments, the payloads may include at least part of the first code 200 and/or second code 202. For example, the payload 204 may be a string representing all or part of the first code 200. In some embodiments, payload may refer to an output of a function or code. The payload may be, but is not limited to, a string, one or more paragraphs, HTML code, and/or one or more structured data objects. Additionally, the payloads 204, 206 may refer to part of an entire payload. In some embodiments, the payload or part of the payload being compared will contain text or HTML content.

[38] The control function 208 and the candidate function 210 may be evaluated in any order, or in a random order. For example, the control function 208 may be evaluated before the candidate function 210, or the candidate function 210 may be evaluated before the control function 208. In some embodiments, the control function 208 and the candidate function 210 are evaluated simultaneously and/or in parallel. Both the candidate function 210 and/or the control function 208 may be idempotent. In some embodiments, the output of the control function or the candidate function is only returned and/or used after the comparison has been completed.

[39] Once the first and second payloads have been generated, a comparison of the first payload and the second payloads is performed at 130 and 135. At 130, the method iterates over at least one pair of nodes. The at least one pair of nodes includes a first node from the first payload and a corresponding second node from the second payload.

[40] The comparison of the first payload 204 and the second payload 206 is performed by iterating over at least one pair of nodes at 130, the at least one pair of nodes including a first node 212 from the first payload 204 and a second node 214 from the second payload 206. In some embodiments, the first payload 204 relating to the first code 200 is directly compared to the second payload 206 relating to the second code 202. In some embodiments, only a portion of the first payload 204 is compared to a corresponding portion of the second payload 206 relating to the second code 202. For

example, a section of the first code 200 and the second code 200 may be selected for comparison, such that only pairs of nodes within the selected section are compared. A plurality of sections from the first code 200 and the second code 202 may be selected for comparison. The plurality of sections are not required to be adjacent or sequential within the code. Comparison of a portion of the first payload 204 and the second payload 206 can increase the speed with which the comparison is performed, as the comparison function only needs to iterate over the selected nodes, rather than iterating over all nodes in the entire code. This may be useful for iterative code modernisation, allowing sections of code to be updated or modernised in stages. In some embodiments, multiple iterations of comparisons are performed for each individual section of code to be compared.

[41] The at least one pair of nodes includes a first node 212 from a first payload 204 and a corresponding second node 214 from the second payload 206. The nodes may be predefined. In some embodiments, the nodes may represent defined portions of a payload. A node may be a data structure that stores a value of any data type and/or a node may be a data type. A node may contain a pointer to another node. For example, nodes may include lists, arrays, strings, objects tables, table headers, table cells, templates, commands, loops, paths, unions, structures, tuples, maps, stacks, queues, graphs, and/or containers. A node may be defined by a start and end tag. A node may be defined by a string which represents a data type or data structure, wherein the string represents an HTML fragment. In some embodiments, the nodes are formed from one or more elements. Elements represent individual components of data structure. The elements may represent an individual value, such as a character or integer. For example, a node which represents a string "abc" may be made up of elements "a", "b", and "c", which are character type values. In another example, a node which represents a table header cell may be made up of elements which include the content of the table header cell. In some embodiments, elements may represent a plurality of individual values or structures which form part of a larger data structure such as a table, object or string. Elements which form a node may be of the same data type or have different data

types. Elements may each contain one or more characteristic(s) associated with the element.

[42] Characteristics may include positions, data type, traits, xpaths, classes, styles, classifications, information, metadata, and/or tags. Characteristics in text may be retrieved using the character's Unicode category, and may include, but are not limited to Unicode categories defined by the Unicode Standard, such as ClosePunctuation, ConnectorPunctuation, Control, CurrencySymbol, DashPunctuation, DecimalDigitNumber, EnclosingMark, FinalQuotePunctuation, Format, InitialQuotePunctuation, LetterNumber, LinesSeparator, LowercaseLetter, MathSymbol, ModifierLetter, ModifierSymbol, NonSpacingMark, OpenPunctuation, OtherLetter, OtherNotAssigned, OtherNumber, OtherPunctuation, OtherSymbol, ParagraphSeparator, PrivateUse, SpaceSeparator, SpacingCombiningMark, Surrogate, TitlecaseLetter, and/or UppercaseLetter.

[43] In some embodiments, traits provide a programmatic comparison of a string to a regular expression. That is, traits are descriptions of particular patterns, and/or can be used to search for particular patterns or occurrences of defined characters and/or character combinations within a string. In some embodiments, traits may be used in the code to describe an html node, or may be used to describe characteristics of text (in a similar manner to Unicode categories). In some embodiments, the term trait may be substituted for "attribute" or "characteristic" of the text. In some embodiments, traits may be used to classify or categorise all or part of a string. The traits may be predefined to match a particular pattern, subset, and/or combination of regular expressions. In some embodiments, the regular expression or regex defined node traits are used to categorise and/or describe information about patterns in text to assist in determining the type of content of a node from an obscured log. The regular expressions may include and/or be defined by Unicode properties. In one example, a trait may be defined as "punctuation", where "punctuation" matches any kind of punctuation character. In another example, an "email" trait may match any character combination having a format of one or more characters on each side of an "@" character. In a further

example, the trait “line-breaks” may be matched to the presence of a carriage return character and/or a new line or line feed character appearing within a string. Examples of traits include, but are not limited to, empty, non-empty, leading space, trailing space, non-breaking space, words, line breaks, punctuation, ampersand, email, and/or number.

[44] At 135, individual elements 218, 220 within the at least one pair of nodes are compared by iterating over each of the individual elements to identify elements between the two nodes which do not match. The comparison may be performed by a compare function. Each element in the first node 212 is mapped to a corresponding element in the second node 214, and a direct comparison of each element is performed. For example, a first element 222 in a first node may be determined to correspond to a second element 224 in a second node if they are determined to have the same or similar attributes and/or position within the node. In some embodiments, elements within the first node 212 and second node 214 may be mapped or determined to correspond if they are nested within the same tags or structured objects. The first element 222 and the corresponding second element 224 are then compared to determine whether the elements match, or whether they differ in some aspect. In some embodiments, elements are determined to be a match when they are the same in every aspect, that is, they are determined to be an exact match. In some embodiments, elements may be determined to match when they meet a specified similarity threshold, or they may be determined to match when they are determined to be equal across one or more values. For example, the character “a” and “A” may be determined to be a match upon comparison based on position and value, even though they have a different character class (lowercase vs. uppercase).

[45] **Figure 2** is a schematic flow diagram showing the comparison of first code 200 with second code 202 by comparing a pair of nodes 216. The first code 200 is fed through a control function 208 to generate a first payload 204. The second code 202 is fed through a candidate function 210 to generate a second payload 206. Within the first payload 204 there is at least one first node 212 which corresponds to at least one second node 214 in the second payload 206. Nodes 212 and 214 form the pair of nodes

216. The method iterates over this pair of nodes 216 to perform a comparison by stepping through elements 218, 220 of the first node 212 and the second node 214. Elements 218 of the first node 212 are mapped to corresponding elements 220 in the second node 214 and are iteratively compared to determine differences between each individual element within the elements 218, 220 of the pair of nodes 216.

[46] As shown in **Figure 3A**, a comparison is performed over a pair of nodes 304a, 304b from a first payload 300 and a second payload 302 to identify a differing element. A first node 304a from the first payload 300 is compared against the corresponding second node 304b from the second payload 302 by iterating over individual elements in the groups of elements 312a, 314a within the first node and groups of elements 312b, 314b within the second node 304a to identify elements within the nodes which do not match. In the example shown in Figure 3A, the elements within each of the nodes 304a, 304b have been separated into two groups of elements 312a, 312b and 314a, 314b. For example, each group of elements could represent table content, elements within a specific HTML tag or groups of strings.

[47] In **Figure 3B**, which is a schematic diagram of the mapping of the groups of elements, individual elements within the groups 312a, 314a of node 304a are compared with individual elements within the groups 312b, 314b of node 304b. A difference is identified between the elements 316a and 316b of the element groups 314a, 314b in nodes 304a and 304b respectively, which defines a differing element. Once the comparison function has iterated over each element to identify any differing element(s), the comparison function then moves on to comparing the next pair of nodes 406a, 406b in the payloads 300, 302. The comparison function continues to iterate over each pair of nodes in the payloads until there are no further pairs of nodes over which to iterate.

[48] In some embodiments, an HTML element from one node may be compared with another HTML element from the corresponding node, wherein the HTML element may include a plurality of different data types or classifications. In some embodiments where groups of elements are mapped and compared, comparison of groups of elements

may include individually iterating over each individual element within the group of elements.

[49] At 140 of Figure 1, after comparing each element within the pair of nodes, at least one differing element 226 within the at least one pair of nodes is identified. According to some embodiments, the differing element may comprise an element in each of the pair of nodes, the identified differing element being a pair of corresponding elements that do not match. In some embodiments, the identified differing element may include an element from a first node that does not have a corresponding element in the second node. In other words, the differing element is an element in the nodes in which a difference occurs. A differing element may be identified when a difference occurs between a first element 222 of the first node 212 and a corresponding second element 224 in the corresponding second node 214. In Figure 2, the differing element 226 is identified from the elements 222 and 224. In the example of Figure 2, the elements 222 and 224 are shown to have a different visual appearance for exemplary purposes only. However, differences between elements may include one or more characteristics in value or value type that do not match between two corresponding elements. The differing element 226 may be extracted from the nodes and stored external to the payloads 204, 206. In some embodiments, the differing element 226 may contain pointers or references to the position of the elements 222, 224 in each of the first node 212 and second node 214 respectively. In some embodiments, the differing element 226 may be represented by an object or a variable, or may include a reference to the elements 222, 224.

[50] At 150 of Figure 1, in response to identifying the at least one differing element, a mismatch object for differing element 216 in the pair of nodes is generated, the mismatch object defining at least one characteristic 228 of the differing element 226 of the pair of nodes 216. The at least one characteristic includes at least a position of the differing element in the first node and the second node. In some embodiments, the characteristic may include the value type of the differing element in the first node and the second node. Referring again to Figure 2, a mismatch object 230 is shown to be

generated for differing element 226 in the pair of nodes 216, the mismatch object including the characteristics 228a and 228b of the elements that form the differing element 226. The characteristics 228a correspond to the characteristics of element 222 from the first node 212 and the characteristics 228b correspond to the element 224 from the second node 214.

[51] In some embodiments, in response to identifying the differing element 226, a compare function may generate the mismatch object 230. In some embodiments, the compare function may return a Boolean value. A mismatch object 230 is generated for each pair of nodes which include a differing element. In some embodiments, the mismatch object 230 is generated for each differing element within each pair of nodes. For example, a pair of nodes 224 may have one or more differing element(s), and an individual mismatch object 230 may be generated for each of the individual differing element(s). In another example, the mismatch object 230 may be generated for a single pair of nodes on the basis of multiple differing elements within the pair of nodes. In some embodiments, the mismatch object 230 may be generated to define characteristics of a differing element which represents a group of elements, for example, a string comprised of a plurality of characters.

[52] Once a differing element 226 is identified, a mismatch object 230 is generated for the pair of nodes in which the differing element 226 occurs that defines at least one characteristic 228 of the differing element 226. The at least one characteristic 228 relating to the differing element 226 in each of the payloads 204, 206 is stored in the mismatch object 230. The mismatch object 230 thereby defines at least one characteristic 228 of the at least one differing element 226 related to the at least one pair of nodes 216. In some embodiments, one or more characteristic(s) 228 are defined in the mismatch object 230 to describe the differing element 226. In some embodiments, only the position characteristic is defined in the mismatch object 230. That is, it is possible for the mismatch object 230 to define that a differing element exists at a specific location or position within the payload, without describing the

nature of the difference, and/or without revealing the value(s) of the differing element 226.

[53] At 160 of Figure 1, the mismatch object 230 may be aggregated to a data set. In some embodiments, the mismatch object 230 may be aggregated to a log data set that represents a log of all mismatch objects which resulted from a comparison of the first code 200 and second code 202.

[54] **Figure 4** is a block diagram of an example system architecture which performs a method of comparing a first code 200 and a second code 202, according to some embodiments. The system 400 comprises one or more processors 402, and memory 404. The processor(s) 402 may include integrated electronic circuits that perform calculations, and may include a microprocessor, for example. Memory 404 may comprise both volatile and non-volatile memory for storing executable program code, or data. Memory 404 comprises program code (for example modules or engines) which when executed by the processor 404, provides the various computational capabilities and functionality of the system 400. The memory may include a control function module 406, a candidate function module 408 and a setup module 410. The setup module 410 may be configured to setup the system for comparison of the codes. Setup may include setting up one or more additional modules. For example, setup may include setting up an error handler, a results publisher 412, mismatch publisher 414, and/or a metrics publisher 416 in the system. In some embodiments, the setup module 410 is configured to provide instructions to a compare function module 416, to set up how the first payload and the second payload are compared.

[55] Example code to set up a system to carry out the method of comparing a first code with a second code is provided below:

```
Public string FirstCode (string SomeInput) {  
    Return new Apparatus<string>("ComparisonMethod")  
        .control(() => FirstCode(SomeInput))  
}
```

```
.AddCandidate("candidate", () => SecondCode(SomeInput))
.AddSetup(apparatus => Configure(apparatus, SomeInput))
.Run();
}
```

[56] In the above example, a code path called *FirstCode* is converted and returns an HTML fragment as a string. This string is set as the control code. A second code, *SecondCode*, is added as a candidate code named "candidate". The candidate code refers to code which is intended to be compared with the control code. The candidate code may include changes, regressions and/or updates from the control code.

[57] The control function module 406 may be configured to execute the control function 406. The control function may be configured to receive first code 200 and generate a first payload. The candidate function module 408 may be configured to execute a candidate function. The candidate function module 408 is configured to receive a second code 202 and generate a second payload. In some embodiments, the control function module 406 and/or the candidate function module 408 may be configured to carry out or perform additional functionalities to process, filter or interpret first code 200 and/or second code 202. For example, the control function module 406 and/or the candidate function module 408 may be configured to identify and/or process specific sections of the first code 200 and/or the second code 202 which have been pre-selected for comparison.

[58] The compare function module 416 may be configured to execute a compare function. In some embodiments, the compare function may include one or more compare functions, for example, a plurality of compare functions which are configured to parse and compare a variety of different types of nodes and/or elements from a first payload and a second payload. The compare function may include a code analyser. The code analyser may be in the form of an analyser which parses code in a particular programming language and groups portions of the code into specific nodes. The code analyser may iterate over a plurality of pairs of nodes simultaneously and/or in parallel.

[59] For example, the compare function may include a HTML analyser 418 that parses HTML and groups content into nodes, such as by tag and/or attribute name. In some embodiments, the compare function may include a text analyser 420. The compare function may further include an object analyser, such as a structured object analyser, that may be configured to break down structured objects into readable nodes.

[60] The HTML analyser 418 is used to compare pairs of nodes which represent HTML code fragments. In some embodiments, a string comparison could be used to compare HTML code fragments. An HTML analyser advantageously is able to compare HTML fragments while taking into account the structure of the code itself. The HTML analyser 418 performs a deep comparison by iterating over pairs of nodes which represent portions of the code. The HTML analyser 418 may be configured to normalise HTML entities, for example character references such as *&*; and may be configured to deal with whitespace. The HTML analyser 418 is configured to identify differing elements between pairs of nodes. For example, it may identify elements with different tags, attributes in the wrong order, and/or attributes with different values. The HTML analyser 418 is able to extract and return location information, such as paths and positions, of the differing elements within the pairs of nodes that don't match.

[61] The text analyser 420 is configured to compare text data structures. The text analyser 420 may be configured to iterate over corresponding character pairs in the first payload and the second payload and look for differences in value, type or classification. A character transform function may be used in conjunction with the text analyser 420 to process whitespace before a comparison is performed. For example, it may collapse or ignore whitespace within the nodes and/or elements that are being compared. In some embodiments, the HTML analyser 418 is used together with the text analyser 420 such that the text analyser parses values from the HTML analyser 418. In some embodiments, the HTML analyser 418 may parse values or results from the text analyser 420. The text analyser 420 may be in the form of a nested function within the HTML analyser 418.

[62] The system may further include a results publisher 412. The results publisher may be in communication with a metrics publisher 416 and/or a mismatch publisher 414. The results publisher may be configured to publish the data set and/or log messages. In some embodiments, the results publisher may be configured to interpret the information from the mismatch objects and output a log data set as an error log.

[63] **Figure 5A** illustrates a visual representation of the identification of a differing element 504, and the characteristics 506 which are extracted to generate a mismatch object 508, according to some embodiments. In this example, the differing element 504 occurs between two nodes 500a, 500b in an element at location [1] in the nodes. The element 502a of the first node 500a is an integer data type, visually represented by a stripe pattern, with a value of “1”. The corresponding element 502b of the second node 500b is a character data type, visually represented by a dot pattern, with a value of “a”. The elements 502a, 502b where the difference occurs is identified as differing element 504. Once the comparison identifies differing element 504, it extracts at least one characteristic 506 of the differing element 504 from the elements 502a and 502b. This characteristic may include at least the positions 506a, 506b of the elements 502a, 502b within nodes 500a, 500b respectively. The position of the differing element may include its position within the first payload and the second payload and/or within a particular node. In some embodiments, the comparison extracts the data type characteristic relating to the differing element from the first node and the second node. For example, the characteristics which are extracted from the differing element 504 exemplified in Figure 5A could be the location [1] in each of the nodes 500a, 500b at which the differing element occurs, which may be useful to indicate that there is a difference between the two nodes without revealing any specific information about the nature of the difference, or the exact value of the difference.

[64] **Figure 5B** illustrates a visual representation of the identification of a group of differing elements, and the characteristics 516 which are extracted to generate the mismatch object 518, according to some embodiments. In this example, the differing element 514 occurs between two nodes 510a, 510b in the elements at locations [1] to

[3] in the nodes. The second to fourth elements 512a of the first node 510a are an integer data type, visually represented by a stripe pattern, with a value of “123”. The corresponding second to fourth elements 512b of the second node 510b is a character data type, visually represented by a dot pattern, with a value of “abc”. The elements 512a, 512b where the difference occurs is identified as differing element 514. Once the comparison identifies differing element 514, it extracts one or more characteristic(s) 516 relating to the differing element 514 from the elements 512a and 512b. These characteristics 516 may include the positions 516a, 516b of the elements 512a, 512b within the nodes, and may further include the value types 516c, 516d of the elements 512a, 512b, for example characteristic 516c may indicate an integer type data value and characteristic 516d may indicate a character type data value. In some embodiments, the characteristics 516 may further include the character type or character class, for example, the characteristics 516 may indicate whether the character is a lowercase letter, uppercase letter, numeral or special character. However, none of the extracted characteristics in the mismatch object 518 reveals the actual value of the elements 512a, 512b.

[65] **Figure 6** is a diagram which illustrates a comparison of at least one pair of nodes in a first payload 600 and a second payload 602. A first node 604a comprises elements 606a in the form of a string of 10 characters in the first payload 600 and a corresponding second node 604b comprises elements 606b in the form of a string of 10 characters in the second payload 602. The first node 604a and the second node 604b are compared. The first node 604a string has a value “abcdefgif”, and the second node 604b string has a value “abcdefgHij”. Each character in the first node 604a is an element which is individually mapped and compared against a corresponding character element in the second node 604b. The element 608a representing character “h” which appears in position [7] of the first node 604a is different to the element 608b representing the character “H” which appears in the corresponding position [7] of the second node 604b. This difference is identified as a differing element 608 of the pair of nodes 604a, 604b.

[66] As shown in **Figure 7**, there is a diagram which illustrates the generation of the mismatch object from the identified differing element 608 after the comparison of nodes 604a and 604b shown in Figure 6, according to some embodiments. Once the differing element 608 is identified by identifying elements 608a, 608b within each of the first node 604a and the second node 604b which do not match, a mismatch object 610 is generated which relates to differing element 608. The mismatch object 610 includes characteristics 611 of the differing element 608. The position characteristic 612a for the element 608a within the first node 604a is extracted, being position [0:7] of the first node 604a. The character classification characteristic 614a, for example identifying that “h” is a lower case alphabetic character, is also extracted as a characteristic of the differing element 608 from the first node 604a. Similarly, the position characteristic 612b for the element 608b within the second node 604b, and the character classification characteristic 614b, identifying that “H” is an upper case alphabetic character, are extracted as characteristics of the differing element 608 from the second node 604b. The character classification characteristic may be defined on the basis of the ASCII value of alphabetic characters. In some embodiments, other classifications may be predefined for character type values and/or other data types to compare differences in elements which have the same data type. The classification may be configured to describe a characteristic of the element without revealing the value of the element.

[67] The extracted characteristics 612a, 614a from the first node element 608a and the second node element 608b of the differing element 608 are then stored in the mismatch object 610. From these characteristics, the mismatch object 610 may be configured to provide a message 616 and/or a description to define the characteristics 611 of the differing element 608. For example, the mismatch object 610 may describe the kind of difference which exists in the differing element 608, without specifying anything about the value of the differing element in nodes 604a, 604b. In some embodiments, the mismatch object 610 may use the extracted position characteristic 612a, 612b and classification characteristics 614a, 614b to generate the message 616. For example, based on the example method in Figures 6 and 7, the message 616 may

read: “*Contents differ: texts differ at control 0:7[LowercaseLetter], candidate:0:7 [UppercaseLetter]*”. Another example of a mismatch message may be: “*Contents differ: texts differ at control:2:5 [DecimalDigitNumber], candidate:2:5 [LowercaseLetter]*”. In some embodiments, the mismatch object may also contain metadata about the differing element within both the first payload and the second payload, including the location (line and column) and the xpath of the differing element.

[68] The mismatch object 610 may then be aggregated to a data set 618. The data set may be in the form of a log data set, which represents a log of all mismatch objects generated. A plurality of mismatch objects 610 may be stored in the set 618 to form a mismatch log. The log data set 618 may provide a plurality of messages 616 generated by the mismatch objects 610 which describe the differences between the first payload 600 and the second payload 602. The mismatch objects 610 may be aggregated to the log data set 618 immediately after they are generated. For example, as the comparison is performed and iterates through the pairs of nodes, once a differing element 608 is defined and a mismatch object 610 is generated, the mismatch object 610 is aggregated to a log data set 618 in the order in which it has been generated. This enables a sequential ordering of the mismatch objects 610 as they appear in the first and second payloads 600, 602. In some embodiments, the mismatch objects 610 may be stored temporarily after generation until the comparison is complete, at which point they are all aggregated to a log data set 618 simultaneously. For example, mismatch objects may be stored in a server or database before being aggregated to the log 618. In some embodiments, mismatch objects 610 themselves may be classified based on type of mismatch, and ordered within the log data set 618 on that basis. For example, a mismatch object 610 that indicates a difference of character classification, for example a lowercase character and an uppercase character, may be given lower priority than a mismatch object that indicates a difference in data type, for example a character and an integer.

[69] An example log message from a mismatch object is as follows:

```
{
  "Message": "Contents differ: texts differ at control:2:5
[DecimalDigitNumber], Candidate:2:5 [LowercaseLetter]",
  "Control": {
    "Line": 1,
    "Column": 188,
    "Xpath": "/table[class=someclass'][1]/thead[1]/tr[1]/td[5]/
#text[1],
    "Traits": ["non-empty", "punctuation"]
  },
  "Candidate": {
    "Line": 1,
    "Column": 188,
    "Xpath": "/table[class=someclass'][1]/thead[1]/tr[1]/td[5]/
#text[1],
    "Traits": ["non-empty", "punctuation"]
  }
}
```

[70] Some embodiments provide a number of distinct technical advantages. The methods of comparison described herein are able to thoroughly and safely test the new code in production by comparing the payloads of the first code and the second code. The method is able to return the first payload or the second payload, and can determine whether to return the first and/or second payload based on the mismatch objects generated. In some embodiments, the first payload or the second payload is only returned and/or used after the comparison has been completed. In some embodiments, particular characteristics may be flagged as low priority or low relevance at the time of catching and/or returning the error or mismatch, and may be used to determine whether to return the first and/or second payload. For example, in some embodiments, the mismatch objects generated in a log may be classified or characterised as low priority, such as differences between uppercase letters and lowercase letters or differences in the type and/or number of whitespaces, and may result in the second payload being returned. In other embodiments, where mismatch objects are classified as high priority, such as differences in values or value types, this may result in the first payload being returned. In some embodiments, where the first payload and the second payload are

substantially the same, and/or when there is an absence of mismatch objects, the second payload may be returned.

[71] The compare function also enables comparison by iterating over pairs of nodes, effectively breaking down the code to perform accurate comparisons, which is an effective way to perform like-for-like replacement of functionality of legacy code. Some embodiments of the compare function described herein provide improved methods of performing deep, structured comparison of HTML. The ability of the systems and methods to describe substantially all differences in Unicode Category improves accuracy when applied to code modernisation where maintaining parity is particularly important. Further, the method aggregates one or more mismatch objects to a log data set, enabling the log to return only the mismatches for the differing elements of the payload, rather than returning entire payloads or raw results.

[72] The generation of mismatch objects also provides a distinct technical advantage over existing systems which log raw results and values. The mismatch objects generated by the methods described herein define one or more characteristic(s) of the differing elements of the at least one pair of nodes, rather than defining the differing elements themselves. This is particularly advantageous, as it means that after comparing the payloads, the identified elements are described rather than reproduced. The methods provide structured data that can be logged in a structured format, thereby generating structured logs as the methods aggregate the mismatch objects into a data set. These structured logs are more easily queryable and smaller than raw results, and are easier to look at to identify differences.

[73] Describing the differences, such as where the mismatch is located, or the nature of the difference (for example, the data value type is different), also mitigates the risk of confidential data being leaked since no specific values are reproduced in the data logs. The mismatch objects provide enough information to determine why the comparison failed without exposing potentially confidential or sensitive information. It provides a thoroughly-tested way to safely omit information about the difference and,

in some embodiments, provides greater control over knowing where sensitive and confidential information is located, and by omitting specific values from the error logs, avoids having to identify or flag whether there is sensitive information included in the logs or not. This may be particularly important for compliance with data regulations.

[74] This is an improved way to compare first code and second code. Unless an error occurs in the legacy code itself (that is, the control function), an error in the experiment should not interrupt the user at all. The use of a guarded setup method is also beneficial in both setting up the configuration of the comparison and publishing the log results. Additionally, by providing an error handler to errors, exceptions thrown from a candidate function, during setup or while publishing results will be caught by the systems and methods described herein.

[75] It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the above-described embodiments, without departing from the broad general scope of the present disclosure. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

CLAIMS:

1. A computer-implemented method for comparing a first code and a second code, comprising:
 - generating a first payload from the first code;
 - generating a second payload from the second code;
 - performing a comparison of the first payload with the second payload by iterating over at least one pair of nodes, the at least one pair of nodes including a first node from the first payload and a corresponding second node from the second payload;
 - identifying at least one differing element of the at least one pair of nodes in which a difference occurs; and
 - in response to identifying the at least one differing element, generating a mismatch object for the pair of nodes, the mismatch object defining at least one characteristic of the at least one differing element of the at least one pair of nodes, wherein the at least one characteristic includes at least a position of the differing element in the first node and the second node.
2. The method of claim 1, wherein the method further includes returning the first payload and/or the second payload.
3. The method of claim 1 or claim 2, wherein the method further includes aggregating the mismatch object to a data set.
4. The method of claim 3, wherein the method includes determining whether to return the first and/or second payload based on mismatch objects in the data set.
5. The method of any one of claims 1 to 4, wherein an error is thrown if a mismatch object is generated for a pair of nodes.

6. The method of any one of claims 1 to 5, wherein the first code is parsed through a control function.
7. The method of any one of claims 1 to 6, wherein the second code is parsed through a candidate function.
8. The method of any one of claims 1 to 7, wherein the first code and the second code are idempotent.
9. The method of any one of claims 1 to 8, wherein performing a comparison of the first payload and the second payload includes applying a comparison function over each pair of nodes.
10. The method of claim 9, wherein the comparison function includes an analyser.
11. The method of claim 9 or claim 10, wherein the comparison function includes at least a HTML analyser function configured to compare HTML fragments.
12. The method of claim 11, wherein the HTML compare function is configured to normalise HTML entities.
13. The method of any one of claims 10 to 12, wherein the comparison function includes a text analyser function configured to compare text.
14. The method of claim 13, wherein the text analyser function is configured to apply a character transform function to process whitespace.
15. The method of any one of claims 1 to 14, wherein the at least one characteristic includes a value type of the differing element in the first node and the second node.
16. The method of any one of claims 1 to 15, wherein the at least one characteristic includes an xpath of the differing element in the first node and the second node.

17. The method of any one of claims 1 to 16, wherein the at least one characteristic includes traits of the differing element.
18. The method of any one of claims 1 to 17, wherein the at least one characteristic includes information about a character class of the differing element.
19. The method of any one of claims 1 to 18, wherein the at least one characteristic includes information about a Unicode category of the differing element.
20. The method of any one of claims 1 to 19, wherein the at least one characteristic describes the nature of the differing element in the first node and the second node.
21. The method of any one of claims 1 to 20, wherein the value of the differing element does not form part of the at least one characteristic.
22. The method of any one of claims 1 to 21, wherein the mismatch object defines a log message.
23. The method of any one of claims 1 to 22, wherein the mismatch object includes a message describing the at least one characteristic of the at least one differing element for each of the first node and the second node.
24. The method of any one of claims 1 to 23, wherein the mismatch object includes an output of the differing element in the first payload and an output of the differing element in the second payload.
25. The method of any one of claims 1 to 24, wherein the data set is a log data set.
26. The method of claim 25, wherein the method includes outputting the log data set as an error log.

27. The method of any one of claims 1 to 26, wherein the differing element includes at least one of: elements with different tags, elements with different values, and elements in a different order.

28. The method of any one of claims 1 to 27, wherein the method further includes converting an HTML fragment of the first code to a string before generating a first payload.

29. The method of any one of claims 1 to 28, wherein the method further includes applying a setup function to configure comparison of the first payload against the second payload.

30. The method of any one of claims 1 to 29, wherein the setup function is configured to instantiate at least one of an error handler and a publisher.

31. Non-transient computer-readable storage medium storing instructions which, when executed by a processing device, cause the processing device to perform the method of any one of claims 1 to 30.

32. A computing device comprising:
the non-transient computer-readable storage medium of claim 31; and
a processor configured to execute instructions stored in the non-transient computer-readable storage medium.