SYSTEM AND METHOD FOR RISK, WASTE, AND OPPORTUNITY EVALUATION, ANALYSIS, AND RECCOMENDATION

Technical Field

The present invention relates to a computer-implemented method and system for applying a class library of rules to a collection of data and more specifically a multi-level organization (possibly hierarchical and nested) of rules to enable evaluation, management, modification, analysis, and recommendations of software development and delivery processes and behaviors to identify waste, risks, and increase efficiency.

Background

Modern businesses typically require a number of projects in order to function profitably. Depending on the specific area of business and type of project, a company may need numerous employees and contractors, facilities, software, tools, support, information processing capabilities, and communications systems to complete a specific task within a project. Software development in particular has many complex processes that are performed from beginning to completion. Development, maintenance, and replacement of software may be done in distinct phases or the software can be updated in incremental steps, where design, construction, and development may occur simultaneously. Workplace environments such as these can lead to various errors, as well as excessive waste in time, processes, and resources, especially when under heavy time constraints. Companies also miss significant opportunities of improvement having tunnel vision for completing the task at hand. Companies have avoided this all too common scenario by hiring more employees to monitor these happenings or contracting with outside consultants. This however is costly, which can be especially damaging for fledgling startup companies. Currently, there are software applications that provide evaluation and analysis of the actual computer language while comparing the computer language to a set of rules but these applications do not extend to the mechanics of the entire workplace where there is usually a significant amount of waste, opportunities for improvement, and risk.

Summary

It is an object of the present invention to provide a computer-implemented method and system to visualize the software delivery process of any given solution (model) using collected data and generate textual analyses and recommendations for changes associated with the model with approximate cost, time, resource, and model-based savings and potential improvements.

It is an object of the present invention to provide a computer-implemented method and system to generate textual analyses and recommendations for changes associated with the project with approximate cost, time, resource, and model-based savings and potential improvements.

It is also an object of the invention to provide a computer-implemented method and system that facilitates managing productivity.

It is still a further object of the invention to provide a computer-implemented method and system that substantially ensures “optimal efficiency” in a desired operation.

It is also an object of the invention to provide a computer-implemented method and system that enables users to visually represent models by implementing powerful and simple diagramming methods and reports.

Brief Description of Drawings

Fig. 1 depicts an overview flowchart for the Waste, Opportunity, and Risk Engine.

Fig. 2A depicts a sample of a data dictionary.

Fig. 2B depicts a sample set of rules

Detailed Description

In the Summary above and in this Detailed Description, and the claims below, and in the accompanying drawings, reference is made to particular features of the invention. It is to be understood that the disclosure of the invention in this specification includes all possible combinations of such particular features. For example, where a particular feature is disclosed in the context of a particular aspect or embodiment of the invention, or a particular claim, that feature can also be used—to the extent possible—in

combination with and/or in the context of other particular aspects and embodiments of the invention, and in the invention generally.

The term “comprises” and grammatical equivalents thereof are used herein to mean that other components, ingredients, steps, etc. are optionally present. For example, an article “comprising” (or “which comprises”) components A, B, and C can consist of (i.e., contain only) components A, B, and C, or can contain not only components A, B, and C but also contain one or more other components.

Where reference is made herein to a method comprising two or more defined steps, the defined steps can be carried out in any order or simultaneously (except where the context excludes that possibility), and the method can include one or more other steps which are carried out before any of the defined steps, between two of the defined steps, or after all the defined steps (except where the context excludes that possibility).

The term “at least” followed by a number is used herein to denote the start of a range including that number (which may be a range having an upper limit or no upper limit, depending on the variable being defined). For example, “at least 1” means 1 or more than 1. The term “at most” followed by a number is used herein to denote the end of a range, including that number (which may be a range having 1 or 0 as its lower limit, or a range having no lower limit, depending upon the variable being defined). For example, “at most 4” means 4 or less than 4, and “at most 40%” means 40% or less than 40%.

When, in this specification, a range is given as “(a first number) to (a second number)” or “(a first number) – (a second number),” this means a range whose limits include both numbers. For example, “25 to 100” means a range whose lower limit is 25 and upper limit is 100, and includes both 25 and 100.

An exemplary system for implementing aspects of the subject matter described herein includes a general-purpose computing device in the form of a computer. A computer may include any electronic device that is capable of executing an instruction. Components of the computer may include a processing unit, a system memory, and a system bus that couples various system components including the system memory to the processing unit. The system bus may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such

architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus, Peripheral Component Interconnect Extended (PCI-X) bus, Advanced Graphics Port (AGP), and PCI express (PCIe). It will also be appreciated that embodiments of the invention described herein may be comprised of one or more processing units and unique stored program instructions that control the one or more processing units to implement, in conjunction with certain networks and non-processor circuits, some, most, or all of the functions of method and system described herein. The non-processor circuits may include, but are not limited to, wireless transceivers, network communication modules, signal drivers, clock circuits, power source circuits, databases, and user input and computing devices. As such, these functions may be interpreted as steps of a method to perform method and system for evaluating and analyzing workplace data. Alternatively, some or all functions could be implemented by a state machine that has no stored program instructions, or in one or more application specific integrated circuits, in which each function or some combinations of certain of the functions are implemented as custom logic. Of course, a combination of the two approaches could be used. Thus, methods and means for these functions have been described herein. Further, it is expected that one of ordinary skill, notwithstanding possibly significant effort and many design choices motivated by, for example, available time, current technology, and economic considerations, when guided by the concepts and principles disclosed herein will be readily capable of generating such software instructions and programs with minimal experimentation.

Aspects of the subject matter described herein are operational with numerous other general purpose or special purpose computing system environments or configurations such as server computers, hand-held or laptop devices, multiprocessor systems, microcontroller-based systems, set-top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, personal digital assistants (PDAs), gaming devices, printers, appliances including set-top, media center, or other appliances, automobile-embedded or attached computing devices, other mobile

devices, distributed computing environments that include any of the above systems or devices, and the like.

The system in the present invention is a Waste, Opportunity, Risk processing engine (WORE) comprising, a rules creator, a rules collection, global dictionary, and a module to identify waste, opportunities for improvement, and risks found. The method for the system in the present invention is embodied in Figure 1. The method generally comprises: creating at least one rule wherein the rule comprises at least one test for evaluating one or more instance of data; storing the created rule in the rules collection; creating at least one data element comprised of a at least one key and value in the global dictionary; storing the global data into the global dictionary; implementing the system into a activity data source comprised of at least one instance of data; receiving the data element from the global dictionary wherein the information may be compared to one or more instance of data; receiving an instance of data corresponding to the data source; evaluating one or more instance of data using activity processing or against one or more data element using at least one condition and determining whether to generate at least one conclusion based on the evaluation, each conclusion indicating that at least one criteria is met; determining whether to perform at least one action based on at least one generated conclusion; typically waste, opportunity for improvement, or risk, and error codes may be displayed by the system if there is chance for finding possible enhancement in the process.

In some embodiments the system may be implemented into a .NET Framework but may be implemented into other types of language that are capable of supporting data structures or objects, collections, and robust string processing. The .NET Framework provides language interoperability for several languages, for example versions of C, Java, and Visual Basic, by providing a virtual machine environment. A Common Language Infrastructure (CLI) may translate code in different languages to a Common Intermediate Language (CIL). The CLI is implemented using a Common Type System (CTS) and a Common Language Runtime (CLR). The CLR provides a virtual machine environment to relieve application developers from programming to a specific central processing unit (CPU) and handles system tasks such as memory management and garbage collection. The CTS has a specification that defines all data types and

programming constructs supported by the CLR and the mechanisms for their interaction. The CTS allows the .NET Framework to support the exchange of types and object instances between libraries and applications written using any conforming .NET language.

The system may provide a collection of rules, wherein the rules may be embodied by any suitable means wherein they may perform functions against input data when implemented into a data file. The rule in the preferred embodiment may indicate a risk, waste, or opportunity for improvement, or any combination. A rule is compromised of at least an expression wherein the expression is a string of conditions to test data against, a text value wherein the text value may be HTML based wherein the text value describes a risk, waste, opportunity of cost associated with the expression that may be displayed if the expression is true, and a risk category wherein the risk category is a numeric code indicating the type of risk. Categories of risk indicated by the system may be but are not limited to cost, cultural, customer satisfaction, model, planning, policy/regulatory, quality, scope, team satisfaction, technical, and unforeseeable risks. Unique risks within a category may also be indicated by the system. The text value may have imbedded information extracted from the data source, to allow customization of the text appearance specifically curtailed to an end user. Other optional fields may be added for a particular rule such as level of suggestion, enumerated list of risk severity, and an enumerated list of waste associated with said rule.

If one or more conditions are evaluated, and if it evaluates to true, the rule engine initiates one or more actions. The action may be a displayed text or in some embodiments categorization in the form of graphs or tables. The conditions may be comprised of a data element, a Boolean connectors comprising one or more clauses, and value to compare with. The Boolean expression may be in Disjunctive Normal Form. The clauses in the Boolean expression may comprise logical clauses that allow fields to be compared to other fields or user defined values and may be characterized, for example, as a text comparison, a data comparison, or a numeric comparison. In some embodiments, junctors are limited to {AND, OR} and are required for each Boolean expression with multiple clauses. The Boolean expression clauses may be characterized by: a clause type (e.g., logical); a clause junctor (e.g., AND, OR); a

comparison type (e.g., a number, a date, a letter) that designates what is being compared; a left operand type that is located on the left side of a logical operator and that may be a field or a custom value; the logical operator (e.g., <, >, =); and a right operand type that is located on the right side of the logical operator and that may be a field or a custom value.

The data element in the condition is the item to be tested in the preferred embodiment may be a value from the global dictionary or the activities list but is not limited from these sources and in some embodiments data elements from three or more sources may be tested at once. The global dictionary contains objects accessible throughout the system. The global dictionary is comprised of at least one pair of two strings, a key and a value and may be comprised of multiple pairs. They key name may be defaulted to any valid string up to 128 characters but is not limited. The value may be stored as a string but may also be a different data type. If the data type is not supported then a null value may be returned and any expression compared with it will be False. In some embodiments, the key name determines the data type. For example, the name “DT” assumes the value is a date, the name “#” assumes a numeric value, and the name “YN” assumes a yes or no value.

The system is comprised of at least one global dictionary but other global dictionaries may be added to the system. When the system is evaluating an instance of data against data elements, the system first searches for the data element in the current object and then in the global dictionary. The global dictionary may be searched first by prefixing a term such as “gb” in front of the field name. A rule may be set so that it only searches the global dictionary. The value may also be from the current activity in the process being reviewed. For example, if the source data indicates how long an activity typically takes place at their company and industry standard time is a value in the global dictionary, the test “TypicalTime>CA:IndustryStandardTime” may be used as an expression to analyze if the value is longer than suggested industry standards a text value may appear to the reporting if the time is greater than the industry standard. In this test, CA is the example industry standard time according to the company “CA” but businesses may have their own specifications for this value that may be inserted into the system.

Figure 2A shows a sample of a data dictionary where a set of rules is evaluated to the properties of a .NET project file. The global dictionary may be referenced in a rule by using the syntax “gb.[Key]” wherein “[Key]” is the desired data element to have a rule added. Using this sample the following shows an example of the system, applied to the simple data dictionary wherein there are two rules being evaluated as shown in Figure 2B. The first rule that would report if the debug flag were still on to remind a developer to turn it off before a production release. A second rule would serve as a reminder to make sure the copyright notice contains the current year.

```
Wore.Wore TestWore = new Wore.Wore();
Wore.Rule TestRule1 = new Rule (“gb.DebuYN=true”, “Turn off debugging for production”);
Wore.Rule TestRule2 = new Rule (gb.Copyright~@Year, “Update copyright date”);
TestWore.GlobalInfo.Add(“DebugYN”, “1”);
TestWore.GlobalInfo.Add(“Copyright”, “Copyright @ Clearly Agile 2017”);
TestWore.GlobalInfo.Add(“OutputType”, “Class”);
TestWore.Rules.Add(TestRule1);
TestWore.Rules.Add(TestRule2);
SortedList<string, string> ans = TestWore.SimpleEval();
```

In addition to the global dictionary the system may process a list of activities using activities processing. Activities are a collection of objects, and the system may check the rules against one or more objects in the collection. The activity may be made of any type of object wherein the system may match the properties within the object collection. This may allow for creating a unique set of rules for gathered data. The system may also support functions that operate against the entire activities list. Some of the functions supported may be “COUNT(field:value)” wherein the function returns the number of occurrences of field with indicated value, “SUM(field:value, numeric field)” wherein the function returns the sum of the total value of the indicated field for numeric fields, “PERCENT(field:value,numeric)” wherein the function returns number of times field occurs over number of items in the collection, and “SEQ(field:value)” wherein the function returns the sequence number of the first occurrence of value. In these

functions, the syntax of “field:value” indicates the field name and the value to test the field for.

In some embodiments, the value may be an asterisk wherein any value in the field will be accepted. In some embodiments, the field may contain the percent sign wherein it will be treated as wildcard lookup in the value. Functions such as these allow for decisions to be made about the entire collection. For example, if the collection contains activities in a testing phase, and has properties called “role” and “action,” the following functions may be used: “COUNT(Role:DEV)” wherein the function returns the number of developers involved, “SUM(Typical:*)” wherein the sum of the typical time spent among all activities is returned, “PERCENT(Action:TEST)” wherein the function returns the percent of testing activities over all activities in the collection, “SEQ(Action:CHECKOUT)” wherein the function returns the sequence of when the checkout activity takes place. In some embodiments the Percent and Sum functions take an optional second parameter of a numerical field to be used to compute percent or sum, rather than based on counts. The syntax for a function such as these may be “PERCENT(field:value,numericField)”. Activity functions may be used on either side of the expression, so a function such as “SEQ(Action:Checkout)>SEQ(Action:Build)” may indicate that one activity (Checkout) was performed after another activity (Build). Combining the global dictionary and activities processing may help users perform a complete analysis on a business process, technical process, or series of processes.

In some embodiments, the same data element may then be displayed again in the text value that may appear if an expression is true. In the preferred embodiment, this may be done by enclosing the data element with characters such as “{ }”. For example, if you want to risk that testing should be at least 40% of time spent in a particular list of activities and expression for this may be “PERCENT(action:TEST,typical)<gb.TestPercent)”. The displayed text to incorporate these data elements may be: You are only spending {PERCENT(action:TEST,typical),P1} of your time doing testing activities. We recommend at least {gb.TestPercent,P1} of the activities in a quality assurance phase be spent on testing work.

A second parameter may also be specified which may be the formatting parameter of the variable. Formatting parameters that may be used but are not limited to “Fx” for fixed with x decimal places, “Cx” for currency with x decimal places, “Px” for percent with x decimal places, “SD” for Short date, “YYYY” for Four-digit year, “YY” for two-digit year, “MM” for Numeric month number, and “MMM” for 3-Character month abbreviation.
In accordance with some embodiments of the invention, the system may have an error-checking module wherein whenever an error in the system is detected, an error code property is set to an error code and the function returns an empty occurrence of its return value. For example, if there are no rules found while running a “SimpleEval” function method, the error code may be set to specific number and an empty collection would be returned. Same error codes for the system may include: Code 100 for no rules being defined, Code 101 for no data found, Code 200 for Invalid rules found, Code 201 for rules reference non-existent data elements, Code 202 for missing required rule element, Rule 300 for invalid credentials, and Rule 301 for expired credentials.
In accordance with some embodiments of the invention, the system may have a rule creator configured to receive desired rules from an administrator, a program, or an end user. The rule creator may include one or more components that may operate to facilitate entering, displaying, saving, retrieving, editing, deleting, organizing, validating, and/or performing other actions with respect to rules. A rule collection is coupled to the rules creator so as to receive and store information into a class of data. A new rule may be created and added to a rule collection by using a wrapper method. They system may return a Boolean True if the rule was added or a False if the rule was not added. The system may display an error property code to indicate why the rule was not added to the collection. For example, a 200 invalid rule or 202 missing require ruled element may displayed. The system may also have a function wherein the function analyzes the rule and if the rule is appropriate, add it to the collection of rules. A developer may also directly manipulate the rules collection using standard language constructs to clear the rules, remove rules, or add rules.
In some embodiments, a set of rules may be loaded into the system from a JavaScript Objection Notation (JSON) array or to a collection of rule objects. In addition, a party may create a set of named rules, wherein a developer may upload said set into the

system. The syntax for this method may be Boolean LoadRuleSet(JSON String | Name | Collection, AppendRules =false) wherein if the first parameter is a string, the method checks to see if it is a JSON collection or a simple string. If the value is not JSON or is a Name that is not found, the function returns FALSE and the error code set. If the first parameter is a collection, then the collection is added to the rules property. Append Rules allows the system to either clear the rules first (FALSE) or to add the rules to the existing rules collection (TRUE).

In some embodiments, the system has a Parse Rule Set method of checking data wherein the system reads through each element in the rules and returns a list of the expressions, an error code, and a short message indicating an error code if the rule is improperly entered. The syntax for this method may be “Collection<Rule result>= ParseRuleSet()”. This method may allow a developer to check their rules prior to applying the data set. In the scenario where the Parse Rule Set is called before the data is loaded, the rules are only checked for syntax. If the data is loaded, the rules are both syntax checked and field names are checked for validity in both the global dictionary and activities collection.

Using the global dictionary and activities collection, the system may compare rules against the data. In some embodiments, the system may have a “SimpleEval()” method function wherein the method allows a set of rules to be checked against the Global dictionary. This function also allows the system to parse a simple keyed pair collection and report any issues found. The preferred syntax for this method is “Collection<RuleObjects>=SimpleEval()”. Using this method function, each rule in the rule collection will be compared against the data. If the expression returns TRUE, the rule is added to the result collection. When all rules are checked, the matches are returned by this method. If any rules are invalid, the returned collection is empty and the error code is set. A developer may then use the parse rule set method to determine the exact errors.

In some embodiments, the system may have an “EvalActivities” method function wherein the method allows a set of rules to be checked against a collection of objects defined by a developer or another person. If the rules are valid, a list of all the rules, which are matched, will be returned. Syntaxes used for this method function may be

“Collection<RuleObjects>=EvalActivities()” wherein the system may evaluate the existing activities collection and return the matching expressions or “Collection<RuleObjects>=SimpleEval(List<object>)” wherein the system adds an activity list, thus overwriting the existing activities and returning matching expressions. To further detail the process of these methods the steps would be: the system would determine the optimal path, either iterate rules, checking all activities or iterate activities, checking all rules. For each rule, the system compares the expression against the current data object and global dictionary. If a match is found, the rule is added to the result. If a match is not found, nothing is done. The results are then returned when the method is complete. If any errors are found, the rules collection will return empty and the error code is set. A developer may check the exact errors using Parse Rule Set. The steps that are performed when evaluating expressions are emptying out the Boolean truth collection; breaking the expression into a collection of individual expressions and Booleans; processing each individual expression and creating a True or False result, adding the resulting Boolean value to the truth collection; and once all expressions are evaluated, combining them with the Boolean operators to return a final True/False value. For example if the expression is: “gb.Environment=Net && Activity=Checkout && Manual=Yes” the system would break this into three individual rules and evaluate each one. If gb.Environment=NET is True and Activity=CHECKOUT is True and Manual is True, the entire expression is TRUE and the rule is returned. The rules may also be as complex as desired by combining conditional tests with the Boolean operators AND (&&) or OR (||). The general syntax is Conditional test && Conditional test || Conditional test. The expression is typically processed from left to right but operators can be combined using the { } group syntax. For example, the syntax may be Conditional test && { Conditional test || Conditional test } wherein the first conditional test is computed and saved. The next two tests are then computed. If either test is true, the overall result is true. The result then would be combined with the first conditional test using the && operator

In alternate embodiments, the source data may be received into a web-based system or application, wherein the source data comprises structured information and may include, but is not limited to, traditional written documents, vector graphics, e-commerce

transactions, algorithms, object meta-data, server APIs (application program interfaces), etc. Thus, the source document may be any type of document into which data may be captured for later analysis in accordance with embodiments of the present invention. Preferably the source document is received in a predetermined format to enable efficient control of the document and to also enable any document to be received into the system regardless of who created the document, what business they are in, and manner in which they inserted the document. One or more global dictionaries and one or more rules in the web-based system or application then may then be selected manually or automatically to evaluate the source data wherein the system then will generate a response. The response may be a text display, graph, table, or any other output that may categorize, analyze, or provide recommendation for the end user.

Claims

1. A system comprising the following components:

a rules engine wherein at least one rule is created and stored and wherein at least one rule includes an expression wherein the expression comprises at least one test for evaluating one or more instance of data;

one or more data elements comprised of at least one key and value wherein the one or more data element may be compared to one or more instance of data;

wherein the system is implemented into a source of data comprised of one or more instance of data; wherein the data element may be compared to one or more instance of data using one or more rules, wherein the system decides whether to generate at least one conclusion based on the evaluation of the one or more rules, wherein each conclusion indicates that at least one criteria is met; wherein the system determines whether to perform at least one action based on at least one generated conclusion