

FIELD

5 [001] The present invention relates to a multiple prompt natural language to code generation platform.

BACKGROUND

[002] Frontend systems, such as from OutSystems® and Pega®; and Data Management Systems,
10 such as from DataRobot® and Dataiku®, are examples of low code or no code generating platforms that do not start from natural language. As for platforms that start with natural language, such as ChatGPT® and Co-pilot® by Github®, they fall into 2 categories

- a) Prompt to return a sample body of words that best fit to the natural language prompt provided, e.g. ChatGPT®
- 15 b) Autocomplete codes - providing natural language comments and keying in the functions and the autocomplete is provided (which need to be change, e.g. co-pilot® by Github®)

[003] For approaches a) and b):

- i) Single prompts are used which do not take into account operation of the other prompts
- ii) Provided body of code or autocomplete suggestion are sample codes which need to be further
20 customized to be used
- iii) The prompt entities are not precisely used in the generated codes, e.g. variables

[004] The above problems exist because a lowcode / nocode platform understands the context of the language it is generating, knows the precise syntax to use and where variables will be used, but do not start from natural language. Large language models start from natural language but do not understand
25 the context of the technology stack as their source of sample code or autocomplete writing is sourced from the internet and/or open source repositories.

[005] The present application discloses a code generation platform that solves the above problems of lowcode and nocode platforms which do not include the integrating written requirements to the code. The invention uses both natural language as a starting point and understands the context of the
30 technology stack to precisely provide the right syntax, replaces the right variables and/or entities and provides for multiple inputs to understand the context of requirements and provide the link between the written requirements to the code.

SUMMARY OF THE INVENTION

35 [006] According to a first aspect of the present invention, there is provided a code generation platform configured to: analyse a series of specifications, written in natural language, whose implementation

requires determining how data is to be used to achieve one or more objectives of a task; reiterate generation and display of one or more interim statements, each identifying a requirement of the task and facilitating breakdown of the task, until arriving at a final set of statements, each setting out how data called by one or more variables is manipulated, wherein one or more of the final statements serve to deploy a program script to realise an objective of the task; and run the final set of statements to generate code for the program script.

[007] Each of the interim statements may provide a recommendation for user input. The code generation platform may be further configured to determine whether a current iteration provides the final set of statements by testing whether the code for the program script is generatable from the one or more interim statements of the current iteration.

[008] The one or more interim statements of a current iteration may be generated based on one or more of: the task objectives inferred from the analysis of the series of specifications; and selected one or more of the interim statements generated from an earlier iteration. The final set of statements may be based on the selected one or more interim statements generated across the reiteration.

[009] The code generation platform may be further configured to await selection of the one or more interim statements of a current iteration before generating the one or more interim statements of a subsequent iteration. The code generation platform may be further configured to allow editing of one or more of: the one or more interim statements; and the one or more final statements.

[010] The program script may be generated from code templates on which syntax of the one or more final statements maps, the code templates being stored in a repository. The code generation platform may be further configured to display, for each of the one or more final statements, a confidence level of a match to a code template from the repository. The code generation platform may be further configured to perform the mapping using a machine learning technique, such as natural language processing (NLP). The code generation platform may be further configured to, for each of the code templates on which a final statement maps, replace each argument in the code template with a corresponding variable from the final statement that maps.

[011] The code generation platform may be further configured to use an artificial intelligence engine to generate the one or more interim statements and the final set of statements. The artificial intelligence engine may be based on keyword extraction and inference using one or more of NLP, k-nearest neighbours (KNN) and GNN techniques. In addition to the artificial intelligence engine, the code generation platform may be further configured to use an external artificial intelligence engine in response to determining the need to tap externally hosted data to generate the one or more interim statements and the final set of statements. The artificial intelligence engine may be configured to generate the more interim statements and the final set of statements based on data on which the platform has been trained to generate code for program scripts for similar task objectives.

[012] The generated code for the program script may be incorporable with separate code, so that when the generated code and the separate code are compiled create one or more of the following to execute

the task: an API, a batch scheduler and a user activated scheduler. The API is configured to accept user input for the task.

[013] The code generation platform may be configured to introduce indentation into blocks of the generated code in response to detection of presence of functions or loops in the final set of statements to match the required syntax of the generated code.

[014] The series of specifications, the one or more interim statements and the final set of statements are in natural language format. The series of specifications may be in Agile format.

BRIEF DESCRIPTION OF THE DRAWINGS

[015] Representative embodiments of the present invention are herein described, by way of example only, with reference to the accompanying drawings, wherein:

[016] Figure 1 shows application architecture components of a system which hosts the code generation platform of the present application.

[017] Figure 2 shows production frontend compute cloud services of a system which hosts the code generation platform of the present application.

[018] Figure 3 shows graph data nodes used by the code generation platform.

[019] Figure 4 shows various stages in generating code using the code generation platform of the present application.

[020] Figure 5 shows an epic written in business language.

[021] Figure 6 shows inferred subepic content and subsub epic content.

[022] Figure 7 shows selectable user stories supported by the code generation platform of the present application.

[023] Figure 8 shows confidence levels for user stories created by the code generation platform.

[024] Figure 9 shows a sample program script generated by the code generation platform.

[025] Figure 10 shows a script to execute and test an Application Programming Interface (API) endpoint.

[026] Figure 11 shows a graph network for Python code.

DETAILED DESCRIPTION

[027] In the following description, various embodiments are described with reference to the drawings, where like reference characters generally refer to the same features across the drawings.

[028] This application relates to a platform that is configured to received natural language input to support no code or low code outcomes. No code outcomes refer to an exact matches of written natural language input to code. Low code outcomes refer to low confidence levels of matches of written natural

language to code. Such low code scenarios require human intervention to inspect and/or update the code to enable their proper execution.

[029] In contrast to low code or no code platforms which provide drag and drop graphical tools with embedded functionalities or prebuilt templates, the present application starts with written requirements.

5 The written requirements begins with a series of specifications, which provides the code generation platform with starting information to understand a task that is to be coded into a computer. That is, the series of specifications provides a written description of a task that is to be coded. The series of specifications is presented to the code generation platform to be simultaneously analysed, i.e. when the code generation platform is initialised, it is configured to receive a plurality of specifications to be
10 concurrently analysed to understand the task. The task is then interpreted and linked to codes.

[030] A task (which in the present application means a project; job or assignment with a goal) may have one or more objectives and one or more requirements. In the process of generating the code for the task, the platform learns the objectives and the requirements of the task. Task objectives are goals of the task or what the task seeks to accomplish, while task requirements are details required for the
15 task objectives to be achieved. For example, to achieve the objectives of a multi-layered task to provide a job recommendation engine for suggesting training courses for a new staff, the requirements may be to provide training input to the engine based on other staff historical selection and profile information. Other requirements may be to update the training data with the training course selected by the new staff for use by the next new staff. The platform obtains these requirements by drilling down. On the other
20 hand, achieving the objective of a single layered task to perform a simple mathematical addition (e.g. $A + B$) would just require the operands A and B.

[031] Advantageously, the code generation platform is configured to accept series of specification which is not immediately implementable for having content where code cannot be readily generated (such content being of “epic” style format, explained later in greater detail). Implementing the series of
25 specifications thus requires determining how data is to be used to achieve one or more objectives of a task and determining the requirements of the objectives. Such implementation involves reiteration to break down the task defined by the series of specification into smaller components, so that the series of specification serves as a starting point or seed for a reiterative process, with the end being to code the task.

30 [032] The reiterative process comprises generating and displaying one or more interim statements that facilitate breakdown of the task, where each interim statement identifies a requirement of the task. The interim statements provide recommendations from which users can select, whereby the recommendations provided by the interim statements of a first iteration are based on analysis of the series of specifications, while subsequent recommendations are provided by one or more of: analysis of
35 the series of specifications; and selected one or more of the interim statements generated from an earlier iteration.

[033] The generation of interim statements is reiterated until arriving at a final set of statements, each setting out how data called by one or more variables is manipulated, where one or more of the final statements serve to deploy a program script to realise an objective of the task. However, depending on the content of the series of specifications, the code generation platform may only perform one round of iteration to arrive at the final set of statements. Variables refer to names given to computer memory locations which are used to store data. The data is retrieved, by referencing their names, and processed to realise a task objective. The processing is directed by a program script whose sequence of instructions is determined by the final set of statements deploying that program script. The code for the sequence of instructions of the program script is generated when the final set of statements are run.

[034] The code generation platform may be used as follows. A user may have a task on hand but may not know what the task entails or how to code the task. The user inputs the task in natural language, providing advantages over platforms that are unable to do so. The platform then provides a list of recommendations for the user to review and select those that achieves the task objectives. The recommendations continue until a list of statements is presented which when executed codes the task. The code generation platform is described in greater detail below in conjunction with the accompanying Figures.

[035] Figure 1 shows application architecture components and Figure 2 shows production frontend compute cloud services of a system 100 which hosts the code generation platform of the present application. Figures 1 and 2 show an implementation where the system 100 is cloud based.

[036] The frontend 102 of the system 100 is a templated application based on MEAN Stack (MongoDB, Express, Angular, Node JS) template. Alternatively, other NoSQL document databases may be used, with the frontend 102 designed using other JavaScript tools.

[037] The frontend 102 may be hosted on a cloud 200 (see Figure 2) having server based suite services 202, such as storage 204, load balancer 206 and cloud computing services 208 that allow communication with a database program 210. However, it is also possible (not shown) for the frontend to run on locally hosted systems.

[038] The frontend 102 has standard foundation services 104 such as login, signup, organization and user management, tasks, email and access control (roles). The code generation platform may be accessed through the frontend 102, to create projects, upload data (not shown) used by the project, generate and deploy code 108 for the created projects, and interact with the generated code.

[039] As mentioned above, code generation begins with the code generation platform receiving a series of specifications seeking to code a task. The code generation platform is configured to support code generation that adopts Agile methodology, which uses an iterative approach that incorporates user feedback at every iteration. Programming of the task is broken into phases with emphasis on continuous updating and improvement from the code generation platform receiving user input on the relevance of recommendations provided at each iteration.

[040] Agile methodology has the code generation platform being able to support content being in “epic”, “subepic” or “stories” format. “Epic” content is a high level description, which when used to define a task may not explicitly state its objectives and requirements, which need to be derived. “Subepic” content is a lower level description compared to “epic” content, so that description which assists to derive meaning from “epic” content may be referred to as being of “subepic” content. “Stories” are a subset of epics. User stories are implementable for having variables, features and/or subfeatures where those matching a corresponding code template with highest confidence being used for code generation. The goal of the code generation platform is to arrive at “stories” because they can be converted into coding script. During iteration, interim statements contain “epic” or “subepic” content, while the final set of statements contain “user story” content.

[041] The code generation platform is configured to accept the one or more of the specifications in natural language, which allows business teams, not necessarily coding proficient, to provide task requirements to the code generation platform in natural language not containing any segments of code.

[042] Details of all supported user stories, along with details of each user story, such as features and subfeatures (e.g. control statements like “retrieve” to get data from a graph) of the programming language on which the code generated by the platform is based; version (an indicator of present code version, with support for code based on outdated versions that are rewritten over time); code templates from which the platform generates code; indentation and outdentation information (to support marking the beginning and end of blocks of code, in accordance with the syntax of the programming language on which the code generated by the platform is based) are stored in a graph database 216. The context of features, subfeatures, indentation and outdentation is in the context of the programming language of the code used by the platform, such as commands. Code refers to lines of code used to implement each user story.

[043] With reference to the graph shown in Figure 3. these details are stored in graph data structure format as nodes 302 connected by relationships 304. The relationships 304 connect nodes 302 for features, subfeatures, version and indentations to code, thereby bridging specifications written in natural language to code. The connected nodes and relationships provide functionality that allows reading of user stories, and in response recommend the closest code to realise a task, followed by the inclusion of indentation and version. During epic to user story inference, described later below with reference to Figure 4, the graph is harnessed, for example, by one or more artificial intelligence engines to recommend subepics or user stories.

[044] Returning to Figure 2, the graph database 216 is a component of the backend 212 of the system 100 and communicates with the frontend 102 using, in one implementation, python.. The backend 212 is hosted on the AWS cloud, using EC2 214. The EC2 214 also holds python codes for the one or more AI engines described below in Figure 4. There are other components like S3 bucket 218 for AI (artificial intelligence) engines used by the code generation platform; and to hold backups and files required or uploaded for execution.

[045] Figure 4 shows various stages in generating code using the code generation platform. The code generation platform provides either a first path 402 or a second path 404 to receive user requirements for coding.

[046] The first path 402 is for input specification in epic format requiring several layers of breaking down before arriving at user story format (see stage 414). The second path 404 is for input specification in epic format that requires less breaking down before arriving at user story format (see stage 414).

[047] Along either the first path 402 or the second path 404, the code generation platform would have analysed a series of specifications for the task to be programmed and established how the series of specifications is to be implemented by determining how data is to be used to achieve one or more objectives of the task. The series of specifications is preferably interpreted simultaneously to allow interaction between task requirements (i.e. each specification is not interpreted in isolation, but the series interpreted concurrently to factor the requirement of one specification having an impact on the requirement of another specification). If it is determined during this analysis that code cannot be generated, the code generation platform is configured to generate and display interim statements, where each identifies a requirement of the task and facilitates breakdown of the task. The interim statements are displayed for user review and selection as to which advance generation code for the task.

[048] The code generation platform is configured to then perform drill down generation by reiterating the generation and display of the interim statements until arriving at a final set of statements, each setting out how data called by one or more variables is manipulated. The final set of statements are in implementable format in that running them results in each final statement deploying a program script to realise its respective task objective. Collectively, the program scripts code all of the task objectives, while this code may also be incorporated with other code to bundle the programmed task into one or more of an API, a batch scheduler and a user activated scheduler. Running the final set of statements then generates the code for each of the program scripts.

[049] In both the first path 402 and the second path 404, the code generation platform determines whether a current iteration provides the final set of statements by testing whether code for program script is generatable from the one or more interim statements of the current iteration. The one or more interim statements of a current iteration are generated based on one or more of: the task objectives inferred from the analysis of the series of specifications; and selected one or more of the interim statements generated from an earlier iteration. The final set of statements are based on the selected one or more interim statements generated across the reiteration. The code generation platform is further configured to await selection of the one or more interim statements of a current iteration before generating the one or more interim statements of a subsequent iteration.

[050] Operation flow for the first path 402 and the second path 404 are described below with reference to a task to code a software project.

[051] The first path 402 caters for non-technical users (such as parties from business) where the project specification on hand may be one or more of epic statements and/or one or more lower level statements providing more detail for one or more of these epic statements.

[052] The code generation platform analyses, at epic stage 406, the input specification and interprets that it is written in high level or epic format, requiring several rounds of break down before arriving at statements that the platform can run to generate code for the software project. The code generation platform breaks down the analysed series of specifications into one or more interim statements of lower or subepic level. The code generation platform uses an AI engine 418 to determine suitable one or more interim statements from being trained on data where code has been generated for similar software projects. If the breakdown cannot be performed because of, for example, the training data 424 not having data on similar software projects, the code generation platform may use an external AI engine 422 to tap externally hosted data to generate the one or more interim statements.

[053] The generated one or more interim statements (either from the AI engine 418 or the external AI engine 422) are displayed at subepic stage 408 as recommendations, where the user is prompted 420 to select the most relevant so that the code generation platform can advance down the first path 402 and code the software project. Alternatively, the user may also be prompted 420 to edit the one or more interim statements to provide more details; or formulate a new specification, so that the code generation platform can return recommendations.

[054] Subsub epic stage 410 (i.e. where the generated interim statements in subepic format of stage 408 are to be broken down to a lower level format, called “subsub epic”) occurs if the code generation platform determines that drill down is required. Subsub epic stage 410 follows a similar methodology to that of epic stage 406, except that output from the subepic stage 408 is analysed, i.e. selected recommendations from analysing the input specification from the epic stage 406; edited recommendations from the subepic stage 408; or new specification input in the subepic stage 408.

[055] In subsub epic stage 410, the code generation platform analyses the output from the subepic stage 408 and interprets that it is still at a level too high to be implementable for the platform to generate code for the software project. The code generation platform breaks down the analysed output from the subepic stage 408 into one or more interim statements of lower level. The code generation platform uses an AI engine 426 to determine suitable one or more interim statements from being trained on data where code has been generated for similar software projects. If the breakdown cannot be performed because of, for example, the training data 428 not having data on similar software projects, the code generation platform may use an external AI engine 432 to tap externally hosted data to generate the one or more interim statements.

[056] The generated one or more interim statements (either from the AI engine 418 or the external AI engine 422) are then displayed as recommendations, where the user is prompted 430 to select the most relevant so that the code generation platform can advance down the first path 402 and code the software project. Alternatively, the user may also be prompted 430 to edit the one or more interim statements to

provide more details; or formulate a new specification, so that the code generation platform can return recommendations.

[057] The subsub epic stage 410 is reiterated, i.e. one or more interim statements are generated and displayed until the interim statements of a current iteration is interpreted to be at a low enough level on which code may be generated, which occurs at user story stage 414. At user story stage 414, the code generation platform has a final set of statements which can be run to generate the code for the software project. Similar to the subepic stage 408 and the subsubepic stage 410, the code generation platform uses an AI engine 434 to arrive at the final set of statements from being trained on training data 436 where code has been generated for similar software projects. Alternatively, or in addition to, an external AI engine (not shown) may be used to tap externally hosted data to generate the one or more final set of statements. The code generation platform also allows the final set of statements to be edited.

[058] The first path 402 provides business users an epic inference tool where users can key in epics 502 and the next two levels: subepic content 602 and subsub epic content 604 (see Figure 6) in business language (see Figure 5) and technical user stories 606 that are tied to this epic. Machine learning models, including but not limited to: NLP techniques are then used to extract the keywords; k-Nearest Neighbours (KNN) and graph neural networks (GNN) techniques for the inference based on the keywords. In addition, a third-party LLM tool (Large Learning Model, e.g. openAI) may be used to augment the inference capabilities, allowing more subepics to be generated if data hosted in the code generation platform is lacking. Using this, users can generate multiple user stories from one sentence, and generate code despite not being able to write stories in a technical manner.

[059] Returning to Figure 4, the second path 404 caters for technical users where the project specification input may also be one or more epic statements. However, in contrast to the first path 402, there is less break down and little or no reiteration.

[060] Additionally, the code generation platform may analyse, at technical epic stage 412, input typically provided by technical users. Technical epic stage 412 uses AI engine 438 trained on data of detailed user stories that could be used to generate code for similar software projects.

[061] The generated one or more interim statements are displayed as recommendations, where the user is prompted (not shown) to select the most relevant so that the code generation platform can advance down the second path 404 and code the software project. Alternatively, the user may also be prompted (not shown) to edit the one or more interim statements to provide more details; or formulate a new specification, so that the code generation platform can return recommendations.

[062] While not shown, the generation and display of the interim statements may be reiterated until arriving at the final set of statements of user story stage 414 which can be run to generate the code for the software project. The code generation platform allows the final set of statements to be edited.

[063] Thus, at the user story stage 414, the code generation platform has one or more final statements for both the first path 402 or the second path 404, which are in user story format. It will also be

appreciated that in addition to the user stories output from the epic stage 412, user stories may be keyed in, chosen from a list 702(see Figure 7), or typed it freeform through the frontend.

[064] As mentioned above, the one or more final statements are convertible to code for the software project. The conversion occurs at template code stage 416, which involves program script for the software project being generated from code templates on which syntax of the one or more final statements maps. These code templates are stored in a repository, for example in EC2 214 (see Figure 2) and the mapping may be facilitated by AI engine 442 using a machine learning technique which includes, but is not limited to, natural language processing (NLP) technique in tandem with retrieval of graph node properties (see Figure 3) via graph cypher 444. Other machine learning techniques also include k-nearest neighbours (KNN) and GNN. With reference to Figure 8, the code generation platform is configured to display, for each of the one or more final statements, a confidence level 802 of a match to a code template from the repository.

[065] For each of the code templates on which a final statement maps, the code generation platform is configured to perform argument replacement 446 where each argument in the code template is replaced with a corresponding variable from the final statement that maps. Illustrating with the following user story:

“As a User, I want to add HighValue and LowValue, and save the results as BothValues”

[066] The argument replacement function extracts HighValue, LowValue and BothValues as variables and replaces the arguments in the following code:

```
*args,results = {variables}
args = (retrieve_number_or_variable(arg,variable_dict) for arg in args)
output = args._next_()
for item in args:
    output += item
variable_dict[results] = output
```

[067] The code generation is also configured to introduce indentation into blocks of the generated code in response to detection of presence of functions or loops in the final set of statements. Illustrating using the following user story with the following content:

```
Try
If
For
While
```

[068] When any of these functions are detected, the code generation platform will create indentation in the generated code to match the required syntax of the generated code. The code generation platform is also configured to insert nested indentation e.g.

```
While
    Try
```

If

For

[069] Examples of statements that effect this feature are “As a User, I want to end for loop”. If the statement is not entered, an error will be flagged during deployment of the code.

5 [070] Figure 9 shows sample program script 902 that is generated when the final statements in template code stage 416 are run. The generated code for the program script may then be incorporable with separate code (not shown), so that when the generated code and the separate code are compiled, the output may be one or more of the following: an API, a batch scheduler and a user activated scheduler. For example, one script 1002 to execute and test an Application Programming Interface (API) endpoint
10 (see Figure 10) to which the user can send commands and interact. Accordingly, the code generation platform allows getting from written language to a coherent body of code that can be interacted with to get meaningful results.

[071] Returning to Figure 4, the AI engines 418, 426, 434, 438 and 442 are based on keyword extraction and inference using machine learning methodology, including but not limited to one or more
15 of NLP, k-nearest neighbours (KNN) and graph neural networks (GNN) techniques. In addition, with various stages (epic 406, subepic 408, subsub epic 410, epic 412 and user story 414) prompting for selection of recommendations that meet user requirements, the code generation platform has multi-prompt capability, in contrast to other code generation platforms that only prompt to review a single recommendation.

20 [072] In another approach, the output of the template code stage 416 may be configured to support enterprise code, such as ABAP code (Advanced Business Application Programming from SAP ® systems). This is achieved by changing the code template base 448. A new Graph network (GNN) 450 (also confer Figure 11) will be used to store the ABAP code and its syntax.. Similarly, the output of the template code stage 416 may be configured to support other codes by use of an appropriate code
25 template base 448, supported by a Graph network.

[073] The code generation platform may be hosted on a single computer server or several computer servers (not shown) having at least one memory to store instructions that when executed cause one or more processors to analyse the series of specifications that provide a written description of the task that is to be coded; reiterate generation and display of the one or more interim statements that facilitate
30 breakdown of the task to arrive at the implementable final statements that are used to generate the code for the task. The memory includes read-only memory (ROM), writable memory, random-access memory (RAM) or other type of dynamic storage device that stores information and instructions for the one or more processors.

[074] The disclosed code generation platform thus provides an end-to-end method to get from
35 business requirements to code or the technology stack using natural language and inference to code capability. Precision to code occurs because natural language is translated for compatibility with syntax of the output technology stack (including, but not limited to, python, ABAP, Oracle and .Net). In

contrast to other code generation platforms that provide sample codes or autocompletes of code, the disclosed code generation platform generates working codes from understanding of the content of user requirements. The disclosed code generation platform also contrasts against other agile software development tool, such as Jira, where the translation of requirement to code requires manual effort, i.e., human programmers, to generate the respective code. The disclosed code generation platform generates code automatically, which reduces or eliminates the need for programmers to write code themselves. This allows simple use cases to be automatically generated on our platform, with the human programmer able to concentrate on more complicated use cases. Additionally, getting from requirements to code supports automatic tracing of requirements to code.

[075] The disclosed code generation platform finds utility by enterprises to generate AI type use cases augment their systems e.g. a recommendation engine for a widget on a digital screen. For example, an e-commerce portal may be augmented with a recommendation engine and a search engine. The recommendation engine can suggest a category and subcategory of a newly added product from their name and description, so that end-users do not have to trawl through a long list to select the category and subcategory they desire. The search engine, similarly, allows potential customers to quickly filter through the range of products on offer to the desired one, providing ease of access.

[076] In the application, unless specified otherwise, the terms "comprising", "comprise", and grammatical variants thereof, intended to represent "open" or "inclusive" language such that they include recited elements but also permit inclusion of additional, non-explicitly recited elements.

[077] While this invention has been described with reference to exemplary embodiments, it will be understood by those skilled in the art that various changes can be made and equivalents may be substituted for elements thereof, without departing from the scope of the invention. In addition, modification may be made to adapt the teachings of the invention to situations and materials, without departing from the essential scope of the invention. Thus, the invention is not limited to the examples that are disclosed in this specification but encompasses all embodiments falling within the scope of the appended claims.

CLAIMS

1. A code generation platform configured to:
analyse a series of requirements, written in natural language in Agile format, whose
5 implementation requires determining how data is to be used to achieve one or more objectives;
reiterate, using artificial intelligence, generation and display of one or more interim
statements, each facilitating breakdown of the requirements, until arriving at a set of tasks, each
setting out how data called by one or more variables is manipulated,
10 simultaneously analyse all the tasks to deploy a program script to realise the one or
more objectives.
2. The code generation platform of claim 1, wherein each of the interim statements provides a
recommendation for user input.
- 15 3. The code generation platform of claim 1 or 2, wherein the platform is further configured to
determine whether a current iteration is final by testing whether the program script is generatable.
4. The code generation platform of any one of the preceding claims, wherein the one or more
interim statements of a current iteration are generated based on one or more of: the objectives inferred
20 from the analysis of the series of requirements; and selected one or more of the interim statements
generated from an earlier iteration.
5. The code generation platform of claim 4, wherein the set of tasks is based on the selected one
or more interim statements generated across the reiteration.
- 25 6. The code generation platform of any one of the preceding claims, wherein the platform is
further configured to await selection of the one or more interim statements of a current iteration before
generating the one or more interim statements of a subsequent iteration.
- 30 7. The code generation platform of any one of the preceding claims, wherein the platform is
further configured to allow editing of one or more of: the one or more interim statements; and the set
of tasks.
8. The code generation platform of any one of the preceding claims, wherein the program script
35 is generated from code templates on which syntax of the set of tasks maps, the code templates being
stored in a repository.

9. The code generation platform of claim 8, wherein the platform is further configured to display, for each of the set of tasks, a confidence level of a match to a code template from the repository.

10. The code generation platform of claim 8 or 9, wherein the platform is further configured to perform the mapping using a machine learning technique.

11. The code generation platform of claims 8 to 10, wherein the platform is further configured to, for each of the code templates on which a task maps, replace each argument in the code template with a corresponding variable from the task that maps.

12. The code generation platform of any one of the preceding claims, wherein the platform is further configured to use an artificial intelligence engine to generate the set of tasks.

13. The code generation platform of claim 12, wherein the artificial intelligence engine is based on keyword extraction and inference using one or more of NLP, k-nearest neighbours (KNN) and GNN techniques.

14. The code generation platform of claim 12 or 13, wherein in addition to the artificial intelligence engine, the platform is further configured to use an external artificial intelligence engine in response to determining the need to tap externally hosted data to generate the one or more interim statements and the set of tasks.

15. The code generation platform of claims 12 to 14, where the artificial intelligence engine is configured to generate the more interim statements and the set of tasks based on data on which the platform has been trained to deploy program scripts for similar objectives.

16. The code generation platform of any one of the preceding claims, wherein generated code for the program script is incorporable with separate code, so that when the generated code and the separate code are compiled create one or more of the following to realise the one or more objectives: an API, a batch scheduler and a user activated scheduler.

17. The code generation platform of claim 16, wherein the API is configured to accept user input for the task.

18. The code generation platform of any one of the preceding claims, wherein the platform is configured to introduce indentation into blocks of generated code for the program script in response to

detection of presence of functions or loops in the set of tasks to match the required syntax of the generated code.

19. The code generation platform of any one of the preceding claims, wherein the one or more
5 interim statements and the set of tasks are in natural language in Agile format.

20. The code generation platform of any one of the preceding claims, wherein the series of requirements is in the form of epic statements and the set of tasks is in the form of user stories.