Software Proposal Document for project Automatic Recognition Of Suitable Design Pattern

Clara Kamal, Farida Mohamed, Hashem Mohamed, Veronia Emad
Supervised By: Dr. Taraggy Mohiy and Eng. Nada Ayman

October 18, 2019

Abstract

According to the English dictionary, Design Pattern is defined as a formal approach of recording a generic reusable solution to a design problem in a specific environment. Software design pattern is considered as one of the most productive discoveries in the software industry since it has a major role in improving the software quality. However, choosing the suitable design pattern for each design problem is not that clear and effortless but requires a good base knowledge about each DP and its functionality. In this system, we aim to produce an automatic approach that supports the suitable selection of a design pattern depending on the requirements of the user using a Question/Answer approach. Moreover, the system will provide the user with the class diagram depending on the selected design pattern. The system will process the user's input through a tree-like model to reach the most accurate selection.

1 Introduction

1.1 Background

Software design patterns (DP) are a set of documented solutions to common software design problems that software developers face through their code construction repeatedly. DP were first popularized by the four authors Gamma, Helm, Johnson and Vlissides [4] that defined 23 design patterns. Therefore, they were known as the Gang of Four (GoF) and were classified to creational, structural and behavioral according to their scope, purpose and their level of abstraction. They are not considered as finished designs to be used directly in a machine code but as ready-made templates. According to [4], each of these templates are documented in a format that holds mainly a "pattern name", "intent", "motivation", "structure", "participants", "known uses", "implementation" and "a sample code".

As maintainability and re-usability are from the main concerns of software engineering life cycle, selection of the suitable DP is considered as one of the most critical and challenging phases since they are reusable object-oriented solutions. They improve the understandability of code, deal with hidden issues that appears in the future, simplify and accelerate the development process. Moreover, the selection of the suitable software DP can cause confusion to software developers because there are several design patterns that may seem similar in their purposes and actions. As an example, there are two design patterns that are object creational design patterns which are Factory DP and Builder DP but the key difference is how the object is being created. Furthermore, using the wrong design pattern which is known as anti-pattern, makes the design pattern worse than using no pattern at all. Therefore, determining the suitable DP needs to identify the scenario and the problem and to have a good knowledge base about each DP and their functionalities in order to make the most efficient and accurate decision.

Background

Design Patterns:	Structure	Function	Context "Send Feedback"	Challenge	Skills "View wirechart"	Signature	Scope
1)Adaptor	Wrap a legacy object that provides an incompatible interface with an object that supports the desired interface	To access a foreign implementation of native functionality	The requirement for concrete type implementation is already met, but by a non-native type	To make the different type substitutable within the native interface	*To initialize Adaptor *To implement The Native Interface	Adapter references (or contains) Adaptee, representing it. Adapter implements Interface. Adapter references (or contains) Adaptee, representing it. Adapter implements Interface	General (Preferably, languages where late binding is obligatory)
2)Façade	Wrap a complicated subsystem with an object that provides a simple interface						
3)Proxy	Wrap an object with a surrogate object that provides additional functionality						

4)Strategy	Define algorithm interface in a base class and implementations in derived classes	To reconfigure discrete functionality	"Dynamic classification": The implementation of a discrete object behavior (typically, less than a method) is determined during its creation and may be reconfigured later *(e.g. formatting, in a configurable environment)*	To encapsulate the specific behavior. To make it easily replaceable	*To initialize context *To preform partially configurable-action	Context references global Strategy. Alternatively, Context contains Strategy (which contains data)	General (In dynamically-typed languages, instance method override may do)
5)Factory Method	Define "create Instance" placeholder in the base class, each derived class calls the "new" operator and returns an instance of itself	To construct objects by fixed type criterion	Creating an object by fixed criterion. Possibly reconfiguring it later	to create the object without specifying its type	*To obtain concrete product *To create product	Factory Method (typically Singleton) creates Products for Client. Each Concrete Factory Method is built to produce the respective Concrete Product. In the extreme case, Concrete Factory Method is parameterized over Concrete Product	Languages that that lack the class object. *(Elsewhere, where objects are created by default, a global reference to the respective class object will suffice)*
6)Visitor	Define "accept" method in first inheritance hierarchy, define "visit" methods in second hierarchy ... a.k.a. "double dispatch"	To assign discrete functionality by object type	Some polymorphic processing is configurable and, therefore, may not be part of the processed type family (e.g. formatting, user interface). Processing depends as much on the processor type as on the processed type and must be done by the processor.	To add the virtual function hierarchy to a class hierarchy without opening it	*To preform configurable operation over heterogeneous collection	Visitor processes Subject by requesting Subject to "accept" it (the Visitor). Concrete Subject, requests Visitor to "visit" it (the Concrete Subject)	Languages that do not support multi-methods *(most commercial OO languages).* *The Visitor is really a multi-method over Concrete Visitor and Concrete Subject types).* Aspect oriented programming attacks the problem by from another direction.

7)Builder	The "reader" delegates to its configured "builder" each builder corresponds to a different representation or target						
8)State	The FiniteStateMachine delegates to the "current" state object, and that state object can set the "next" state object	To switch among alternative implementations of an entire functionality	Dynamic classification": Object that receives a small number of messages, but the entire set of methods it uses to respond to them depends upon its current state	*To encapsulate state management *To virtually replace an object's effective virtual table	*To initialize state machine *To respond to event	FSM contains States, references the current State and interfaces for it	General
9)Bridge	The wrapper models "abstraction" and the wrapper models many possible "implementations" the wrapper can use inheritance to support abstraction specialization	To separate choice of implementation from interface	Two alternative classification schemes seem equally valid (e.g. stream opened for either input or output and also encapsulates a specific device). A replacement for multiple inheritance, possibly with dynamic classification	To base the design on one classification while retaining the latter. Possibly, to allow the latter to vary during object lifetime	*To create configured behavior *To preform concrete action	Behavior contains Implementation. Client uses Concrete Behavior and typically provides the Concrete Implementation	General
10)Observer	The "model" broadcasts to many possible "views", and each "view" can dialog with the "model"	To synchronize state change	"Controlled data redundancy": The state of one object must reflect the current state of another object (e.g. document and its views, server and its clients, the result of a formula and its values in a spreadsheet)	To keep the dependent object up-to-date at minimum cost. (The alternative of polling the data source by its observers is both expensive and intrusive)	*To prepare for subject change *To change subject state	Subject references notifiers, each referencing an observer	General

11)composite	Derived Composites contain one or more base Components, each of which could be a derived Composite	To traverse a recursive structure uniformly	A recursive object structure (e.g. file directory tree) has to be traversed from the outside (e.g. view traversing its document), applying routine functionality (e.g. formatting for display)	*To traverse the structure node by node, ignoring node type (possibly applying operations that affect the integrity of the structure) *To copy or move nodes, ignoring Node type (counting on automatic validation)	*To process heterogeneous subtree	Both particular Leaves and default Composite are Concrete Components. Composite contains (or references) Components. The Particular Composite may choose to limit the scope of the inherited association (see rectangular inheritance of association).	General
12)Decorator	A Decorator contains a single base Component, which could be a derived Concrete Component or another derived Decorator	To enhance discrete functionality dynamically	"Dynamic and multiple classification": The implementation of some facet of behavior may be extended during its object's lifetime	*To encapsulate the difference in behavior without modifying the subject. To make the extended functionality over the subject	*To decorate concrete subject *To use subject	Decorator is a concrete Subject and references (another) Subject, interfacing for it. Concrete Decorator derives from Decorator	General (Preferably, languages where late binding is obligatory)

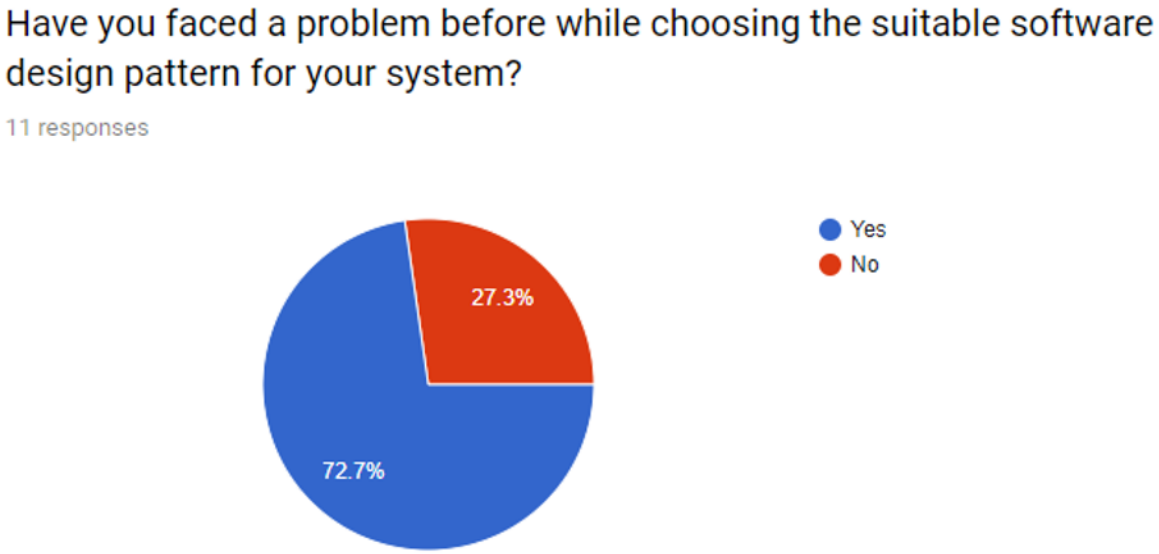
				substitutable with the subject *To practically replace an object base (setting it to an existing object) *To replace super call by delegation			
13)Chain Of Responsibilities	Define "linked list" functionality in the base class and implement "domain" functionality in derived classes						
14)Interpreter	Map a domain to a language, the language to a recursive grammar, and the grammar to the Composite pattern						

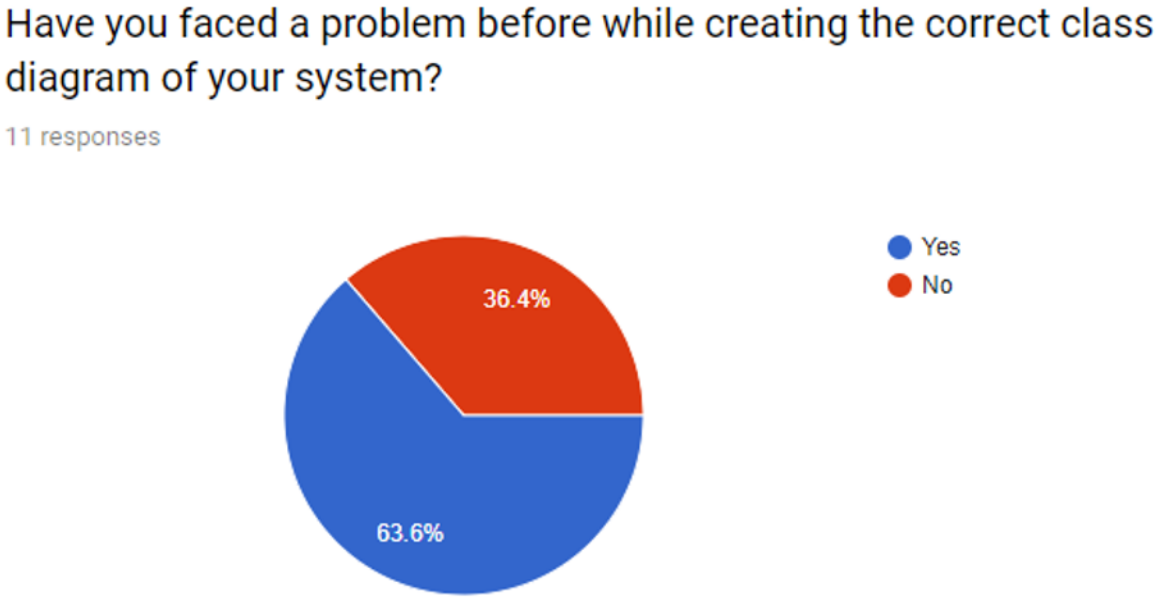
15)Command	Encapsulate an object, the method to be invoked, and the parameters to be passed behind the method signature "execute"	To separate request from execution	Initiator of action cannot (or need not) be there in time to launch the action (e.g. scheduling, selection from menu). Possibly, action may be undone later	To encapsulate the execution request with its arguments (and possibly its undo arguments)	*To create execution request *To invoke execution request	Invoker contains Commands. Concrete Command References Receiver. Client references (or contains) both Receiver and Invoker, creates the Command and registers it with Invoker	Languages that do not support bound methods / delegates. General (when involving more data or functionality than just the receiver, method and invoke-time arguments)
16)Iterator	Encapsulate the traversal of collection classes behind the interface "first..next..is Done"						
17)Mediator	Decouple peer objects by encapsulating their "many to many" linkages in an intermediary object						
18)Memento	Encapsulate the state of an existing object in a new object to implement a "restore" capability						
19)Prototype	Encapsulate use of the "new" operator behind the method signature "clone" ... clients will delegate to a Prototype object when new instances are required	To create objects by example	*Selecting objects by visual example. *Creating objects by content (rather than type). *Copying heterogeneous collections.	To copy each object without having to tell its type	*To prepare example set	prototype responds to *clone* message, returning base object	General *(Preferably, languages that support deep-copying by default)*

20)Singleton	Engineer a class to encapsulate a single instance of itself, and "lock out" clients from creating their own instances	To encapsulate a globally-available resource	*A software module (in languages that do not support modularity) *A generally available part of the programming infrastructure whose existence is taken for granted by the programmers who use it *A non-procedural flow of control implicit in the construction of infrastructure components prior to the main program	*To guarantee the existence of only one singleton object *To complete the construction of the singleton instance prior to its first use	*To assess singleton instance	Added static instance method, static singleton instance, private constructor and possibly destructor	Languages that do not support modularity
21)Abstract Factory	Model "platform" (e.g. windowing system, operating system, database) with an inheritance hierarchy, and model each "product" (e.g. widgets, services, data structures) with its own hierarchy ... platform derived classes create and return instances of product derived classes	To construct objects by criterion and preconfigured type	*Creating objects by two criteria: Base type (Known to the application) and environment (preconfigured). *(E.g. creating GUI controls, given control type and knowing the GUI system being emulated)* *An array of factory methods with the same implementation criterion. *(Same example as above, where the need for each control type has aroused on a separate occasion)*	To encapsulate the decisions in a single object	*To obtain concrete product *To create product	Abstract factory responds to creation messages explicitly named after each conceptual product type. Each concrete factory implements this interface for a concrete environment. The products are arranged in respective discrete type families where environment implementations derive from product	General
22)Template Method	Define the "outline" of an algorithm in a base class ... common implementation is staged in the base class, peculiar implementation is represented by "place holders" in the base class and then implemented in derived classes	To outline and guarantee the execution of a generic process	"inheritance of process": A type family shares a sequential process with one or more stages being type-specific	To avoid repetition of the entire process in all implementations	*To execute a generic process	Processor features the template method (which should not be virtual) as well as one or more "primitives" - virtual (possibly abstract) functions, typically private. Concrete Processor implements the primitives	General
23)Flyweight	When dozens of instances of a class are desired and performance bogs down, externalize object state that is peculiar for each instance, and require the client to pass that state when methods are invoked	To prevent redundant creation of global resources	Objects are heavyweight or encapsulate system-critical resources and are read only	To prevent resource duplication, system wide	*To prevent redundant creation of global resources	Flyweight contains Smart Pointer Counters by key, creates Resources (which it keeps via the Smart Pointer Counters) as well as Smart Pointers (which it does not keep) that share a Smart Pointer Counter. Smart Pointer Counter contains a Resource and notifies the Flyweight. Client uses Smart Resource Pointers, obtained from the Flyweight	General

1.2 Motivation

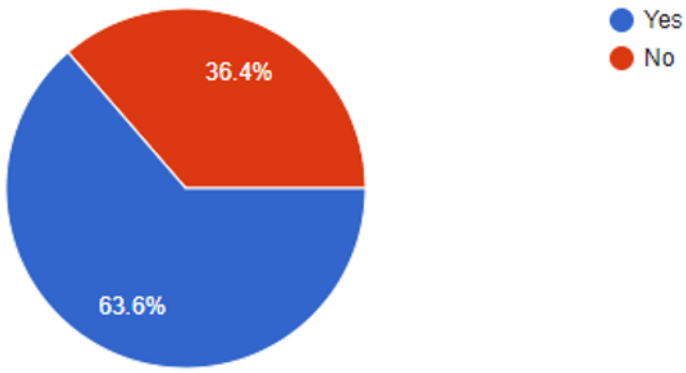
1.2.1 Market Motivation





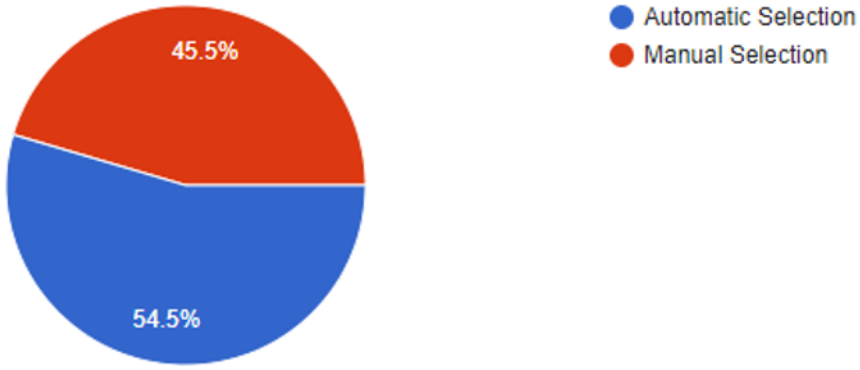
Have you ever discovered after writing your code that you used the wrong software design patterns?

11 responses



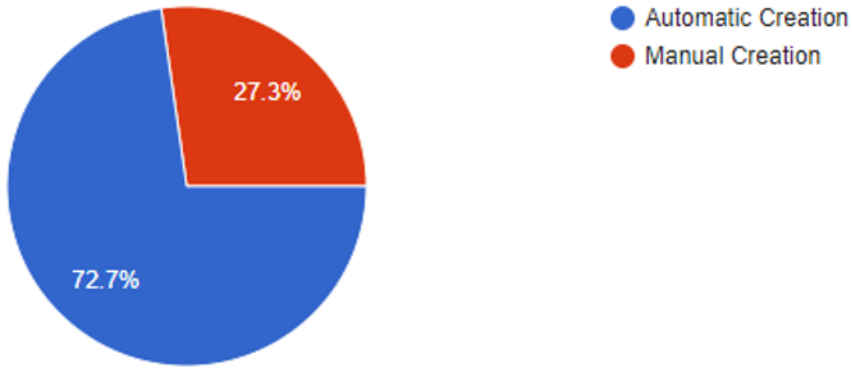
Do you prefer automatic or manual selection of software design patterns?

11 responses



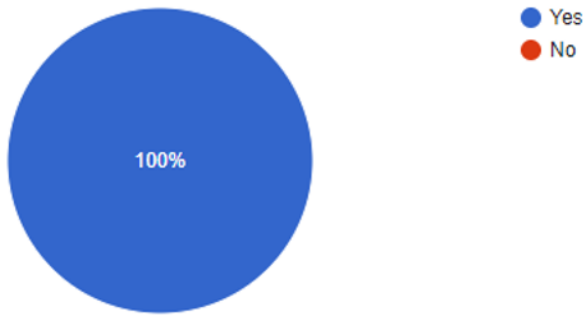
Do you prefer automatic or manual creation of class diagrams?

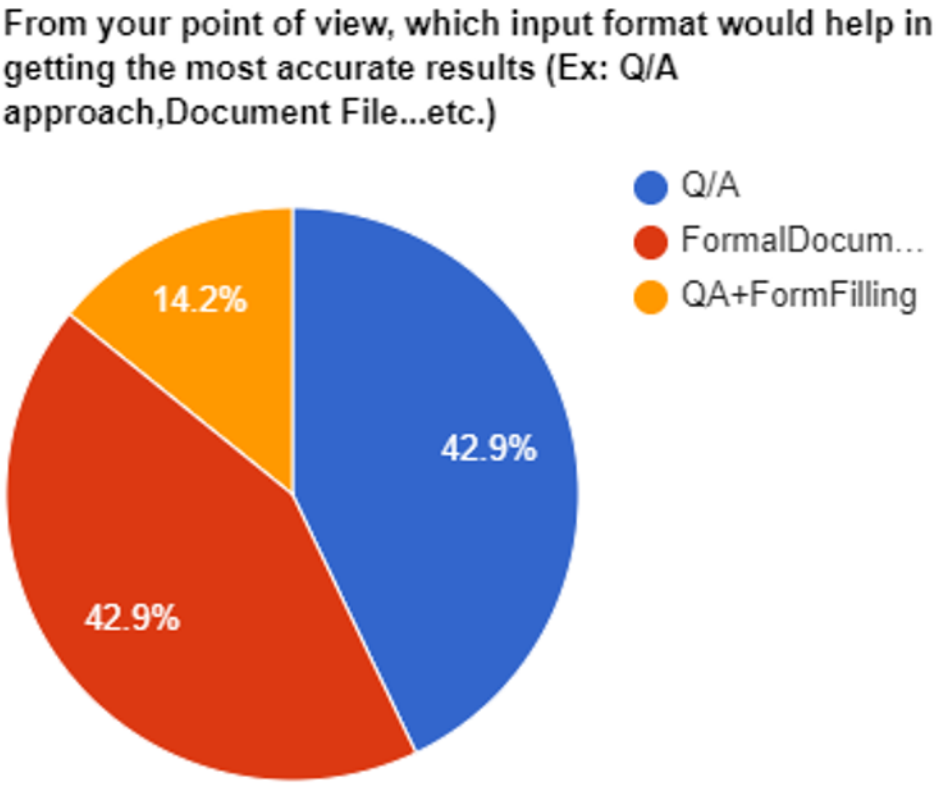
11 responses

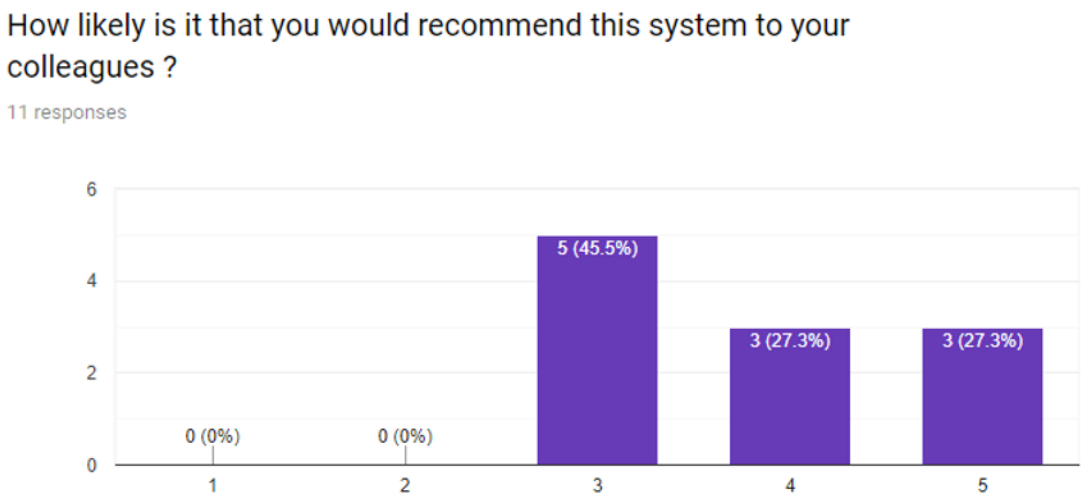


If there's a system that helps in selecting the suitable software design patterns or creating the class diagram, Will you use this system ?

11 responses




From your point of view, which input format would help in getting the most accurate results (Ex: Q/A approach,Document File...etc.)
Q/A
FormalDocum...
QA+FormFilling
14.2%
42.9%
42.9%



1.2.2 Academic Motivation

Design pattern selection is considered one of the most critical and confusing phases of software development.So the purpose of this approach is to determine an efficient accuracy that most of the previous researches didn't reach. This approach aims to aid software engineers in selecting the suitable design pattern despite their knowledge about it.one of the main factors that influence this approach is that there are very limited researches about automatic selection of software design patterns.Also few researches have tested their approach on the most common 23 design patterns which gives our approach more accuracy level.

1.3 Problem Definitions

Software engineers encounter several problems while developing any system . These problems not only affects the performance of the software engineers but also cost them a lot. As an example, Software maintenance can become intolerable and expensive. After designing and implementing a system, software engineers may discover that there are some problems that can let them re-design and re-implement their system as in ignorance and misapplication of appropriate design patterns during the early phases of design and development. Software engineers also encounter communication complication. Because each software engineer has his own way in designing and implementing, which consume a lot of time and effort to explain it to another software engineer working on the same system. Long codes are also one of the main problems that software engineers encounter because they have an extremely poor rate. Software engineers find

it hard to explain their purpose from the code to someone learning this code because of its complexity. Software design patterns comes up with solutions to all the problems that software engineers encounter. They do assist to write more understandable code with useful names for what software engineers are trying to accomplish. They allow the code to be maintained easier because it is more understandable. They help software engineers in delivering design goals amongst other software engineers. They present the intention of their code instantly to others. They allow writing less code because more of the code can derive common functionality from common base classes. Almost all of the design patterns are tested, proven and sound.

2 Project Description

The system aims to provide an automatic selection of the suitable design pattern that is more accurate than the manual selection for the software developers who need help in selecting the design patterns because of their low knowledge about them. In order to provide that, the software developer needs to fill a form of questions that's designed upon the definitions of all 23 design patterns then the system will extract the developer's requirements from his answers. Next, these requirements are processed through a dictionary made of the design patterns documented templates and the weights of the answers are calculated until it reaches a suitable design pattern for the developer and finally generating its class diagram using the plug-in Plant UML.

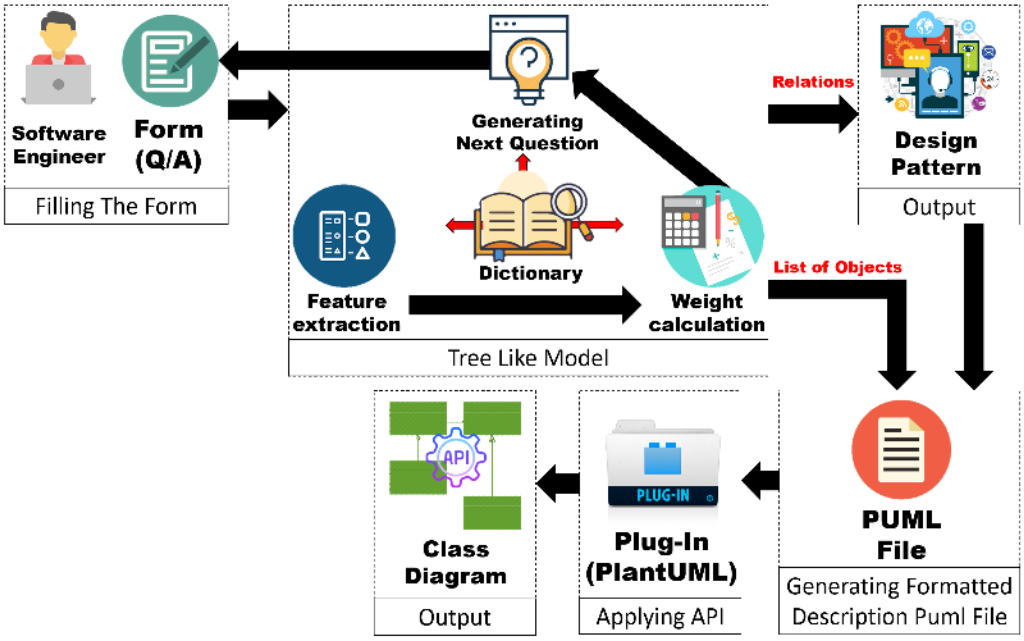
2.1 Objective

The system is developed in a form of an assisting tool for software engineers. It will help them in selecting the suitable software design pattern and create its class diagram according to their requirements accurately.

2.2 Scope

- The system will provide an automatic selection of the suitable design pattern according to the user's specific design problem.
- The system will create a class diagram depending on the design pattern chosen.

2.3 Project Overview



3 Similar System Information

1. User Requirements To Class Diagram Analysis :
   A system proposed by Hatem Herchi, Wahiba Ben Abdessalem.[6] The system proposes an approach to help class diagram extraction from textual requirements using NLP techniques and domain ontology.They used Gate Framework which is an open source framework developed using the Java programming language. It provides a set of natural language analysis tools which can take English language text input and gives as a result the base forms of words, and mention which noun phrases refer to the same entities. It provides an (IE) information extraction system called ANNIE which is (A Nearly-New Information Extraction System) which provides multiple language processing methods as Sentence splitter, Parts of speech (POS tagger) and Syntactic parser. The system has several steps to generate a class diagram . These steps starts with The NL (Natural Language) analysis block processes the requirements descriptions submitted by the user using the framework GATE, and specially Sentence splitter. Candidate classes are extracted by checking the noun phrases in the requirements text. Candidate relationships can be detected in the same way by checking verb phrases. They used domain ontology which supported their system by excluding the unrelated elements, and then preserved the convenient elements to be processed to generate the class

diagram. The system has 83 % Recall and 93% Precision.

2. Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques:
   A system proposed by Meryem Elallaouia, Khalid Nafil, Raja Touahni[7]. This System propose a technique of transforming user stories into use cases .They achieved this by using natural language processing (NLP) techniques, by applying TreeTagger parser. They have obtained precisions between 87% and 98%. The Transformation Process have some features , the first step is preprocessing of text file that contains a set of user stories. Followed by an algorithm that removes all unimportant words. After that the new file is parsed by TreeTagger parser which creates parse tree for each user story, through which noun , proper noun, determiner and verb can be selected.This parse tree helps the extraction of actors, use cases and their associations relationship by the plugin that they have implemented .To obtain UML use cases diagrams automatically from user stories, Their plugin receives only sentences that comply with the pattern proposed by Wautele[4]: As a(n) for the actors, I want to , I can , I am able for the actions, and so that for the benefit.Their Future work will be based on addressing other types of relationship, such as generalization and specialization between used cases and actors.

3. Recommendation System for Design Patterns in Software Development (DPR):
   A system proposed by Francis Palma, Hadi Farzin, Yann-Gael Gueheneuc [8]. The main purpose of this system is that software maintenance can become tiresome and expensive because of using inappropriate design patterns during the initial phases of design and development, which would be determined by recommendation systems for software engineering which can assist designers and developers with several activities that include design patterns suggestions. This System provides a Design Pattern Recommender(DPR) process overview for software design to recommend design patterns, depending on a simple Goal-Question-Metric (GQM)approach. The results that were gained with DPR illustrated that the system can draw a clear conclusion that DPR is more efficient with a few additional features, in comparison to another System as ESSDP, ReBuilder and Recommender System regarding flexible weighting scheme and models refactoring scope.

   The system detected the appropriate design patterns with 50% of the trails, It depended on the knowledge of the user that tested the system. This table explains more about the results,the correct answer was Adapter.

4. Dynamically recommending design patterns:
   A system proposed by S. Smith, D. R. Plante[9]. This system was designed because of that many programmers that have knowledge of design patterns, whether they are rushed to meet deadlines, inexpert in their implementations, or unaware of a certain patterns , pattern implementation

may be unnoticed. This System dynamically search for certain gesture that would aid a programmer by using a specific design pattern and make the relevant recommendations throughout code development.As acclaimed by Dong et al.[3], rather than dealing with source code directly, almost all pattern's search algorithms use some form of average code representation. They used The Abstract Syntax Tree (AST) which is a directed acyclic graph, where each and every node shows a programming element and its children are the elements which belongs to a part of it . They used Brute force algorithm and an another algorithm which is similar to BFS (breadth-first search),Which was used to check for every permutation (rearranging) of the nodes in the graph and whether it matches the anti pattern matrix or not."K-Steps" shrinking was used to work with a less classes at a time. Only the classes that are being modified with those "close" to them, should be inspected for pattern matches.

5. Automatic Recommendation of Software Design Patterns: Text Retrieval Approach and Topic Modelling for Automatic Selection of Software Design Patterns:
   A system proposed by Abeer Hamdy and Mohamed Elsayed [5] for the automatic selection of the fit design pattern (DP). The motivation for this system is that it allows the developers to describe their design problems in natural language and that the task of design pattern recommendation is analog to the text retrieval task. So, it solves the problem of the difficulty in choosing the right DP for a given design problem due to the existence of a large number of DP and to help the novice software engineers. The system used 2 textual datasets made manually, one for the 14 patterns from the catalog of GoF design patterns and the other includes 32 real design problem scenarios collected from various sources. It mainly compares the problem scenario after preprocessing it, with the patterns' distinctive words then detects the suitable pattern by checking the occurrence of the same patterns keywords in the problem scenario (it must include words from the design pattern description). They achieved this by using 7 mechanisms, which are: Tokenization, Noise Removal, Normalization then Porter stemming algorithm for text preprocessing. Indexing and Feature Selection by VSM like unigrams and bigrams to distinguish between the different patterns that have similar keywords. TF*IDF (Term Frequency Inverse Document Frequency) weighing mechanism was also used to enhance the performance of the text retrieval process. At last they used Similarity Measure using Cosine Similarity (CS) to retrieve the suitable design pattern.

   This approach managed to find the right design pattern with accuracy 65.5%. Then it was enhanced later in the same year by 72%, which occurred by proposing a new approach which is (Topics and Unigrams). It was improved the natural language toolkit NLTK for Pre-processing and Genism was used for training the LDA topic model to discover the hidden semantics in a text through relating words with similar meaning and used

the Improved Sqrt-cosine similarity instead of CS.

6. Software design patterns classification and selection using text categorization approach paper:
   A system proposed by Shahid Hussain, Jacky Keung and Arif Ali Khan[10]. The system's main motivation is to automate the suggestion of appropriate design pattern(s) according to an assumed design problem in the design phase of software development and to overcome the multi-class problem and to improve learning precision. This solves the issue of a designer with the lack of experience with design patterns has concerns how to find the best one. The system contribution was a new feature selection method (i.e. Ensemble-IG) using a leveraged scheme IGFSS (Improved Global Feature Selection Scheme). It ensembles the one-sided feature selection method OR (Odd Ratio) and global feature selection method Information Gain(IG) regarding the work-flow of leveraged scheme IGFSS.

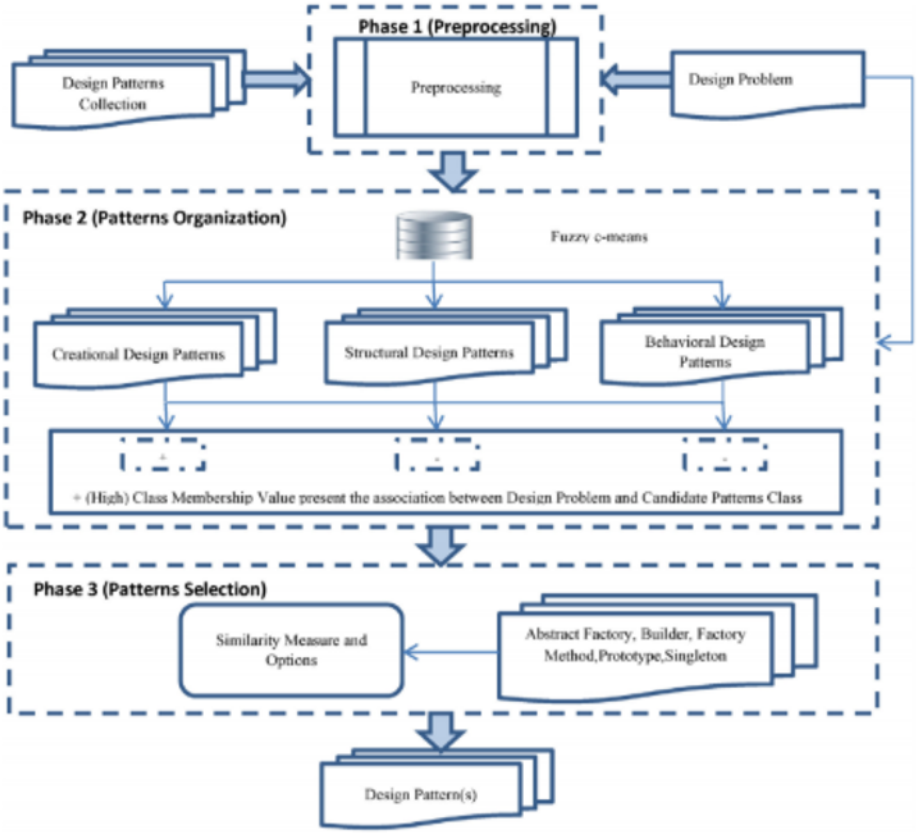
   The dataset included three frequently used design pattern collection, Gang of Four (23 design patterns), Douglass design pattern collection (34 design patterns) Security design pattern collection (46 design patterns).

   The proposed approach has formulated in three phases:

   (a) Preprocessing Using JPreText
      i. Remove frequent words that carry no information
      ii. Word streaming using Porter's stemmer stemming algorithm.
      iii. Indexing Documents and VSM
      iv. Assign Weights to Features (Binary, Term Frequency (TF), Term Frequency Inverse Document Frequency (TFIDF), Term Frequency Collection (TFC), Length Term Collection (LTC), and Entropy)
      v. Feature selection method

   (b) Patterns Organization employed for two purposes:
      i. Group the similar patterns and referred each group as a design pattern class.
      ii. Determine a candidate design pattern class, closer to the description of the given design problem, recommended by Fuzzy c-means.

   (c) Design Pattern Selection using Cosine Similarity (CS).



7. A GQM-based Approach for Software Process Patterns Recommendation: A system proposed by Z. Meng, C. Zhang, B. Shen W. Yin [11] for a recommendation system for the software process patterns based on a Goal-Question-Metric approach. The system consists of 2 phases, Q/A designing phase and pattern recommendation phase. The Q/A designing phase contains 4 steps. First, preprocessing of the process pattern library. Then, building a topic model and get text-topic probability distribution matrix by applying Latent Dirichlet Allocation on all software process patterns scenario descriptions. Next,calculating sentences clustering using K-means algorithm to probability distribution matrix with k=25 and calculating topic similarity using Euclidean distance on vectors in matrix. Finally, designing of the questions from each cluster of pattern descriptions then assigning a right answer to each question on every related pattern. The second phase which is the recommendation of the pattern depends on the answers. It starts by recording the weight of each answer, calculating all the matching degrees between patterns and questions then selecting the top five process patterns. The system designed 57 questions for 89

patterns. The system was evaluated by comparing it with similar systems of the statistics method based on TF-IDF and the evaluation results show that this approach contributes a high F-score which is 11.6% higher than that of the traditional TF-IDF approach. Moreover, the average precision can reach 57%.

8. Software Design Pattern Recognition using Machine Learning Techniques: A system proposed by A. Dwivedi, A. Tirkey, R. Ray and S. Rath [1] has been developed to recognize the design patterns using machine learning techniques. It helps in clearing away the issues occurred in reverse engineering which are false positive and false negative issues using the classification techniques such as Layer Recurrent Neural Network (LRNN) and Decision Tree on Abstract Factory and Adapter patterns. Its first step is to start preparing the dataset used in training by defining the patterns using their template elements and considering their structure and behavior. Next, it defines each pattern classes then constructs object-oriented metrics-based feature vectors. This process is done by providing an open source software, JHotDraw, to a detection tool, such as similarity scoring algorithm, Web of Patterns Metrics and Architecture Reconstruction Plugin for Eclipse, to extract the pattern instances and providing the same open source software to JBuilder tool simultaneously to extract the matrices- based candidate classes. Subsequently, the extracted information will be mapped together to create feature vectors. Finally it starts preprocessing these metrics-based dataset, 80% for training and 20% for testing. The second step is detecting the design pattern by checking if the pattern participants are available in source code are instances of the actual software patterns or not. This process starts by learning the metrics-based feature vectors by concerning LRNN and decision tree classifiers to identify explicit boundaries between classes. Then the recognition of the design pattern is done by checking the composition of pattern participants available in training dataset against learned design pattern participants and ve-fold cross-validation has been performed to remove undertting and overtting from the selected models. Finally the results obtained are validated. This approach used some parameters to evaluate their system and they are precision, recall, F-measure and a confusion matrix to calculate the system's accuracy. As a result of this approach, it was found that this system have got better results of precision than the previous similar systems since the adapter pattern produced 100% accuracy using both classifier while the abstract factory pattern produced 100% accuracy using LRNN, 97.7% using decision tree on the testing dataset and 99.4% using decision tree on the training dataset.

   This system used the reverse engineering approach, known as backward engineering or design recovery process, that depends on reversing the phases of the software creation and starts with taking an existing product as an input. Although reverse engineering takes less time than forward engineering in development, reverse engineering have some risk factors that

makes it difficult to retrieve the original design of the existing product such as the loss of embedded business knowledge and difficulty to retrieve an efficient design and requirements.

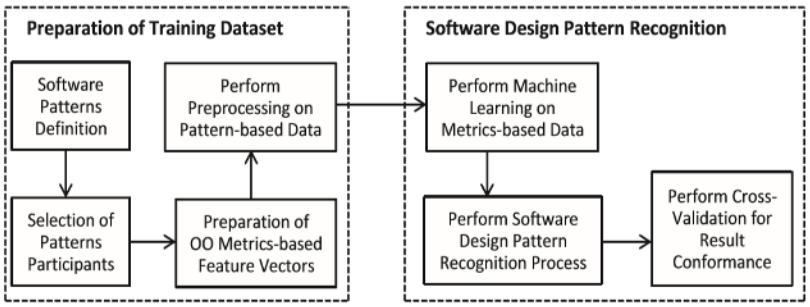


Fig. 3. Presented Software Design Pattern Recognition Model

3.1 Similar System Description

A similar approach to our approach concerning selecting the suitable design pattern is Goal-Question-Metric (GQM) approach. It is an approach to software metric that has been promoted by Victor Basili[2] and Software Engineering Laboratory at the NASA Goddard Space Flight Center. It is based upon the assumption that for an organization to compute in a determined way it must first determine the goals for itself and its projects, then it must trace those goals to the data that are expected to illustrate those goals operationally.It consists of Conceptual level (Goal), Operational level (Question) and Quantitative level (Metric) which is a set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

Another similar systems used different techniques to detect design patterns and create class diagrams automatically. There is a system developed by Hatem Herchi, Wahiba Ben Abdessalem [] which extract textual requirements and generates a UML class diagram from it using NLP and domain ontology. Another similar system is automatic recommendation of software design patterns using text retrieval approach, where the design problem scenarios are illustrated in natural language (NL). A vector space model (VSM) was generated for the catalogue of design patterns. A vector of features composed of unigrams and bigrams are generated for the stated design problem scenario.[4]

3.2 Comparison with Proposed Project

Comparison with Proposed Project

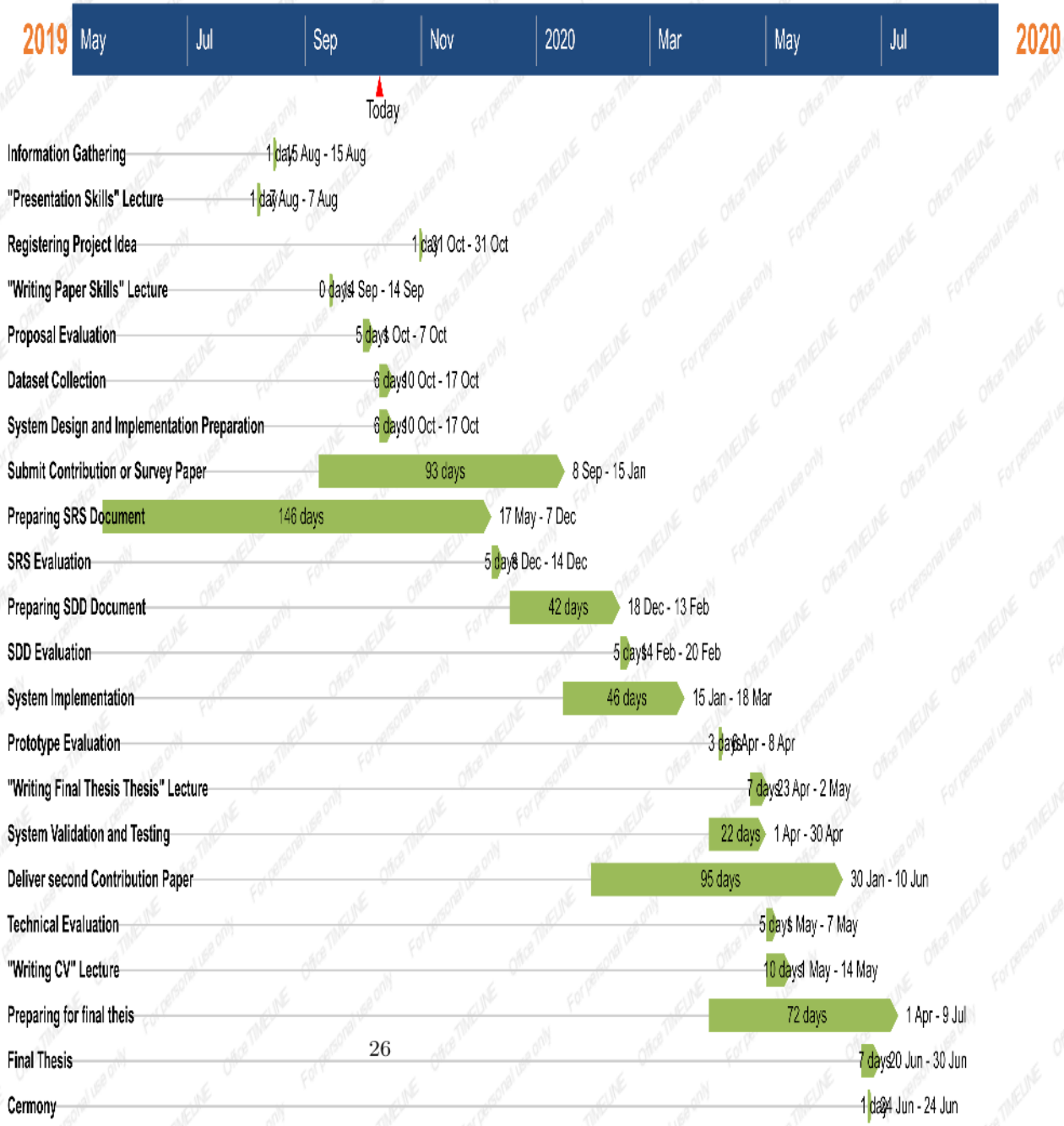
System	Functionality	Techniques	Dataset	Results
From User Requirements To UML Class Diagram	The system proposes an approach to help class diagram extraction from textual requirements using NLP techniques and domain ontology	Sentence splitter, Parts of speech (POS tagger) and Syntactic parser.	Eleven rules proposed by Chen []	The system has 83% Recall and 93% Precision
Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques	This System propose a technique of transforming user stories into use cases	1- Natural language processing (NLP) techniques 2-TreeTagger parser	Unifying and Extending User Story Models, in Advanced Information Systems Engineering by Wautele[]	Thee system has obtained precisions between 87% and 98%
Recommendation System for Design Patterns in Software Development: An DPR	This System provides a process to recommend design patterns, depending on a simple Goal-Question-Metric (GQM)approach.	Goal-Question-Metric (GQM)	E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software.	This system detected the appropriate DP with 50% of the trails, It depended on the knowledge of the user
Automatic Recommendation of Software Design Patterns: Text Retrieval Approach and Topic Modelling for Automatic Selection of Software Design Patterns	Describe the design problems in natural language to choose the appropriate design pattern	Text processing : Tokenization, Noise Removal, Normalization ,Porter stemming algorithm Indexing and Feature Selection by VSM Cosine Similarity (CS) TF*IDF weighing mechanism	14 patterns from the catalog of GoF design patterns and the other includes 32 real design problem scenarios collected from various sources.	This approach managed to find the right design pattern with accuracy 65.5%. Then it was enhanced later in the same year by 72%

Software design patterns classification and selection using text categorization approach paper	automate the suggestion of appropriate design pattern(s) according to an assumed design problem in the design phase of software development	Preprocessing Using JPreText Porter's stemmer stemming algorithm Indexing Documents and VSM Binary, Term Frequency (TF), Term Frequency Inverse Document Frequency (TFIDF), Term Frequency Collection (TFC), Length Term Collection (LTC), and Entropy IGFSS (Improved Global Feature Selection Scheme)	Gang of Four (23 design patterns), Douglass design pattern collection (34 design patterns) & Security design pattern collection (46 design patterns).	This paper mention that they got a higher precision in each technique was used however the percentage wasn't mentioned
Dynamically recommending design patterns	This System dynamically search for certain gesture that would aid a programmer by using a specific design pattern and make relevant recommendations throughout code development	The Abstract Syntax Tree (AST) Brute force algorithm and another algorithm which is similar to BFS (breadth-first search) K-Steps shrinking	Problems to be solved while code development Anti Pattern detection	N/A

Automatic Recognition Of Suitable Design Pattern (Our System)	Our system detects the suitable design pattern using question and answers approach generates UML class using API	Goal-Question-Metric (GQM)	Using our own dataset based on Gang of Four (23 design patterns) as a theoretical reference and other online resources	To achieve higher accuracy To gain independent efficient results

4 Project Management and Deliverables

4.1 Tasks and Time Plan



4.2 Budget and Resource Costs

Budget for a PC

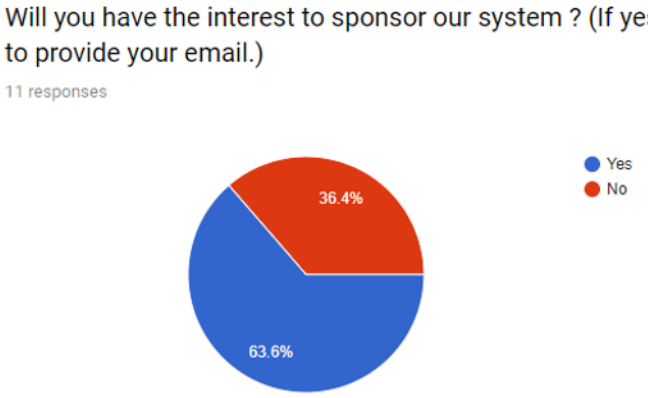
4.3 Supportive Documents

ihabsafwat.18@gmail.com
petergeorge143@gmail.com
omarkhaledzidann@gmail.com
seifhassabelnaby@gmail.com
wassemcs@gmail.com
Saad.adel539@yahoo.com
Hend1410317@miuegypt.edu.eg
ayman@fcih.net
imoustafa@iskydev.com
mhanfy@iskydev.com
minamed7atag@gmail.com

Company Name (the place you work in)

11 responses

Centro global
eBenefits Network
Asset
.
freelancer
Silicon unions
Vodafone
Na
iSkyDevelopment
iSky Development
Valeo



5 References

References

[1] R. R. A. Dwivedi, A. Tirkey and S. Rath, *Software Design Pattern Recognition using Machine Learning Techniques* , 2016.

[2] V. Basili, *Goal Question Metric (GQM) model* , 2006.

[3] D. et al, *A review of design pattern mining techniques* , 2009.

[4] J. Gamma, Helm and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* , 1994.

[5] A. Hamdy and M. Elsayed, *Automatic Recommendation of Software Design Patterns: Text Retrieval Approach and Topic Modelling for Automatic Selection of Software Design Patterns* , 2018.

[6] W. B. A. Hatem Herchi, *Design Patterns: Elements of Reusable Object-Oriented Software* , 2012.

[7] R. T. Meryem Elallaouia, Khalid Nafil, *Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques*, 2018.

[8] Y.-G. G. Palma, Hadi Farzin, *Recommendation System for Design Patterns in Software Development (DPR)* , 2012.

[9] D. R. P. S. Smith, *Dynamically recommending design patternss* , 2017.

[10] J. K. Shahid Hussain and A. A. Khan, *Software design patterns classification and selection using text categorization approach paper* , 2017.

[11] B. S. Z. Meng, C. Zhang and W. Yin, *GQM-based Approach for Software Process Patterns Recommendation* , 2017.