Music Inspired HS Algorithm for determining Software Design Patterns

A.V.Sriharsha
Research Scholar, Department of CSE
S V University College of Engineering
Tirupati, India
avsreeharsha@gmail.com

Dr. A. Rama Mohan Reddy
Professor, Department of CSE
S V University College of Engineering
Tirupati, India
ramamohansvu@yahoo.com

Abstract— ***Harmony Search (HS) is a phenomenon-mimicking algorithm a metaheuristic inspired by the improvisation process of musicians proposed by Zong Woo Geem (2001). Multiobjective, Multiconstratined optimization for determining the Global Optima from several Local Optima is challenging and toughest task, which can be strategically solved by HS metaheuristic algorithm. The capability of the improvisation process to find the optimal solution in the HS metaheuristic is used for determining the software design patterns based on their characteristic weights deduced using many parametric and non-parametric methods. In this paper we attempt to design a framework for determining the optimal design pattern as the solution for the problem narrated in the form of pattern stories. We have evaluated for some complex case studies and the results are affirmative for the statement declaration.***

Keywords- design pattern discovery, optimal design patterns, formal design patterns, harmony search.

I. Introduction

Problems in the real world are complex in their nature, and software development is not a compromise for real big enterprises that host multiple facilities with a great database, network backup emerging as a distributed environment. Such of those are not built over night, or by a wizard style of programming. They need keen understanding of the problem and a strategic design and amply manned teams for coding, testing and simulation. The prototype of such software is the mind-coded tools that gain acceptance of the users' folk and the developers' ambitions. The great software architecture will be the backbone for such assignment, where tricky software engineering techniques are not sufficient. However, software engineering breathes out into a giant methodology of software architecture building, where no one believes that architecture is built. For such problems where architecture needs to be built there is a need of sketching and design work to be done. The design is modularized and distributed to the coders for module development and with many levels of approvals they are united to show up as big software. The problem lies when the need of doing such a job for so many areas, domains and applications.

II. DESIGN AS THE EVOLUTION OF MODELS

All architects, indeed all designers, manipulate models of the system. These models become successively less abstract as design progresses. In Hatley/Pirbhai the reduction of abstraction is from behavioral model to technology-specific behavioral model to architecture model. There is also hierarchical decomposition within each component. The technology of modules is indeterminate at the top level, and becomes technology-specific as the hierarchy develops. The Q2FD performance modeling technique shows stepwise refinement of customer objectives into engineering parameters. As the QFD chain continues, the engineering parameters get closer to implementation until; ultimately, they may represent machine settings on the factory floor. Likewise, the structure of integrated models in software and manufacturing systems follows the same logic or progression.

Many heuristics evolved to compose a design of a pattern that comes with calculating the relationships among the components of fragment of software. The criteria for evaluating a design of a recurring problem of the real world progresses or evolves in the same manner as design models. In evaluation, the desirable progression is from general to system-specific to quantitative. For heuristics, the desirable progression is from descriptive and prescriptive qualitative to domain-specific quantitative and rational metrics.

Design patterns are a technique for documenting solutions to recurring design problems and for sharing design expertise in an application-independent fashion. But directly applying design patterns for the development platform is a complicated task and only the programming language experts have to be educated about the applications of design patterns. All designers need not be programmers of a language or programmers never go into the strategic concerns of design. Thus a software system development needs to be bothered about the design complexity rather the development complexity.

Different approaches, exploiting software metrics, were used in previous works to automatically detect design concepts and operate clones in large software systems.

Formalizing the design pattern mechanisms have evolved to put any design pattern into practice, Christopher Alexander has explained the first forms of design patterns, then clarified with illustrative diagrams and further for software design patterns specific code examples. This format is very informal and hence brings such ambiguity that it is often a matter of dispute whether an implementation conforms to a pattern or not. Furthermore, it is now widely recognized that a poor presentation of patterns can lead to poor system quality and can actually impede maintenance and evolution. The pattern story of the given software design problem is described formally using the mathematical specifications rather with an illustrated sets of diagrams, where the formalization helps to understand the functional and non-functional parameters of the system.

In this paper we are going to discuss the formal specification of the design problems and as the input the optimization problem to find the optimal design solution, all using the formal specifications of design patterns and harmony search algorithm.

III. METRICS USED FOR DRAWING WEIGHTS FOR PATTERNS

A managerial view of software has been in vogue to understand the modularity and weightage metrics. Most of the models of this view are the familiar tools of project management. In addition, management-related metrics that can be calculated from other models are invaluable in efforts to create an integrated set of models.

Some examples include:

1) The waterfall and spiral system development meta-models; they are the templates on which project-specific plans are built.
2) PERT/CPM and related task and scheduling dependency charts.
3) Cost and progress accounting methods.
4) Predictive cost and schedule metrics calculable from physical and behavioral models.
5) Design or specification time quality metrics — defect counts, post-simulation design changes, rate of design changes after each review.

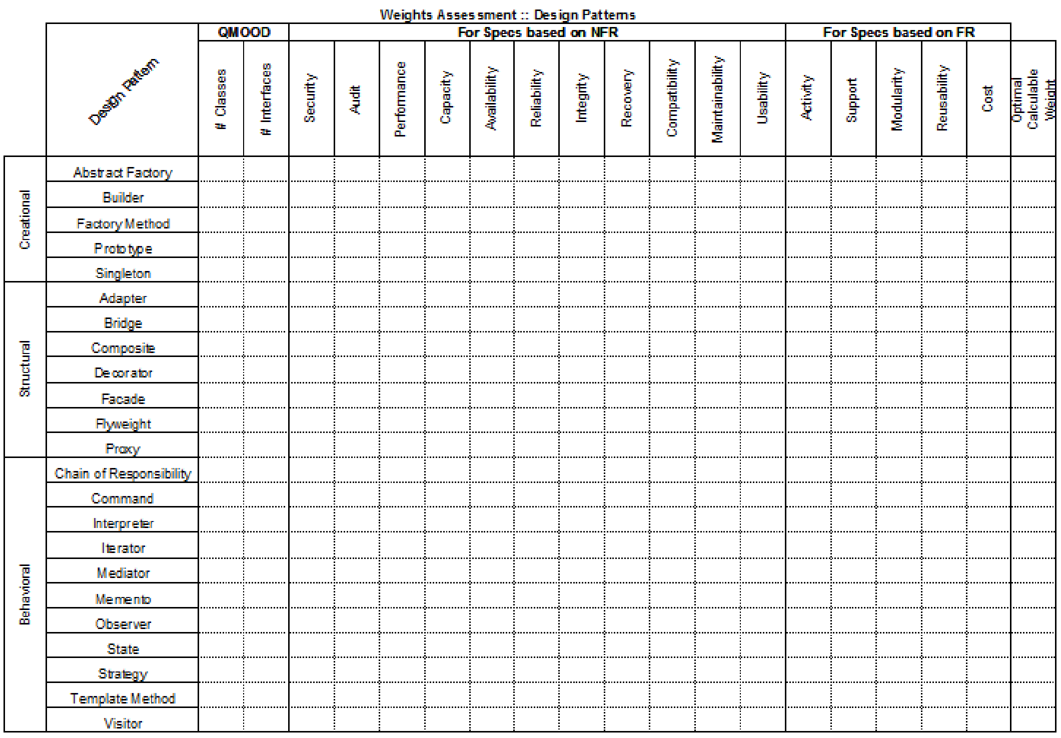
The civil architect will have enough details for proven cost models, and the programmer can measure execution speed, compiled size, behavioral compliance, and invoke quantitative software quality metrics.

Metrics in software engineering are used to evaluate the weight of the developed and running software. They are used to evaluate the volume of the software with various complexity metrics, in terms of LOC, Number of modules and rates of coupling and cohesion. Chidamber & Kemerrer have worked out an agenda for the solutions for determining metrics. Ever since the software have been evolved into Object Orientation the need for measuring the voluminosity of the software in terms of objects, coupling, cohesion and their desiderata of object orientation has become ample importance in the software engineering. Though many programming paradigms have evolved into existence object orientation has become fundamental phenomena for major number of programming languages for their successful domain specific application development tasks.

The formal values for the design patterns had been deduced using the principles of metrics by Quantitative Metrics of Object Oriented Design (QMOOD). This uses evolutionary computation techniques for validating the classes and components, where the same is used to validate the formal values of the design patterns. The formal values calculated for the design patterns when they stay vis-á-vis in the libraries have formal values defined as template formal values, the formal values calculated for the design patterns when they are implemented in the application have formal values as implementation oriented formal values. The design patterns thus have two categories of values one is the template formal values and implementation formal values. The implementation formal values are always more than the template formal values. We would consider the template formal values as the minimum weight given to the design pattern, which is used while searching the design patterns from the library for the user inputs. A design pattern is a collection of well aligned classes, interfaces and polymorphic methods and some ancestors. Each pattern can be visualized as a graph that can be used to analyze specific characteristics of a target software design, such as the number of disjoint inheritance hierarchies. Each metric returns a floating-point number, and those numbers are used as input for formulas that evaluate complex quality characteristics. These characteristics and the formulas that are used to computer them are shown in the table follows:

Descirption	**Name of the Metric**
The number of classes in the software design.	(DISC) Design size in Classes
The number of class hierarchies in the software design	(NOH) Number of hierarchies
The average number of other classes that a class inherits	(NOA) Number of ancestors
The number of methods in the software design that exhibit polymorphic behavior	(NOPM) Number of polymorphic methods
The average number of public methods in a class	(CIS) Class interface size
Counts of the number of classes that a given class is directly related to by attribute declaration or method return type.	(DCC) Direct class coupling
The relatedness among methods of a class, computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class.	(CAM) Cohesion among methods of class
The average number of methods in a class.	(NOM) Number of methods
The ratio of non-public (i.e., private or protected) attributes to the total number of attributes declared in the class. This is interpreted as the average of the ratios for all classes in the software design.	(DAM) Data access metric
The average number of data declarations (e.g., fields) in a class whose data types are user-defined classes. We exclude classes that are part of the Java standard library and language.	(MOA) Measure of aggregation
The ratio of the total number of methods inherited by a class to the number of methods that are accessible by member methods of that class.	(MFA) Measures of functional abstraction

Table 1: Describing the Nature of Parametric Weights of OO Design.



Weights Assessment :: Design Patterns

		QMOOD		For Specs based on NFR											For Specs based on FR					
	Design Pattern	# Classes	# Interfaces	Security	Audit	Performance	Capacity	Availability	Reliability	Integrity	Recovery	Compatibility	Maintainability	Usability	Activity	Support	Modularity	Reusability	Cost	Optimal Calculable Weight
Creational	Abstract Factory																			
	Builder																			
	Factory Method																			
	Prototype																			
	Singleton																			
Structural	Adapter																			
	Bridge																			
	Composite																			
	Decorator																			
	Facade																			
	Flyweight																			
	Proxy																			
Behavioral	Chain of Responsibility																			
	Command																			
	Interpreter																			
	Iterator																			
	Mediator																			
	Memento																			
	Observer																			
	State																			
	Strategy																			
	Template Method																			
	Visitor																			

Fig 1. A Survey Template to acquire the parametric weights of the design patterns

In the above table for each pattern of GoF, the functional and non-functional specifications are weighed and the optimal calculable weight that is the template formal value is calculated, by using QMOOD metrics. Approximate values of certain number of observations of a group of designers have to be attributed here to justify the weights (optimal calculable) here:

The Decorator design pattern:

From the intent of the design pattern Decorator, the following are the intents apprehended:

a) Attaching the responsibilities to the external objects.

b) Recursive-wrapping and support client specified embellishment.

c) Wrapping as a box (a kind of encapsulation and giving protection to the class API).

For a typical and standard application of the Decorator pattern in a word processor kind of application the following metrics are drawn.

Name of the Metric	**Min**	**Max**
(DISC) Design size in Classes	3	5
(NOH) Number of hierarchies	1	2
(NOA) Number of ancestors	1	2
(NOPM) Number of polymorphic methods	3	9
(CIS) Class interface size	2	4
(DCC) Direct class coupling	2	4
(CAM) Cohesion among methods of class	5	10
(NOM) Number of methods	6	26
(DAM) Data access metric	5/12	20/48
(MOA) Measure of aggregation	50	500
(MFA) Measures of functional abstraction	16/50	55/170

Table 2: The parametric weights of Decorator Pattern application on Laboratory Survey.

An example of complex design pattern as template needed to be addressed here: Taking an example for the proof of the algorithmic approach a Decorator pattern is considered. The drawing of the Decorator pattern is advocated to apply for the programming problem and the OO weights of the pattern are evaluated. The design pattern is represented as a tuple of classes and relations among classes. When examining potential pattern instances, to avoid combinatorial explosion in checking all possible class combinations, OO software metrics are used to determine the pattern constituent candidate sets.

Using the above metric the optimal calculable weight for each design pattern that is template formal values are deduced. Parameters that determine the weights are drawn from the procedures of QMOOD and AOL using AST.

IV. HARMONY SEARCH

A meta-heuristic algorithm, that mimics the improvisation process of a jazz music player, has been prominent since a decade and it is named harmony search. This has acquired wide popularity in optimization problems, representing several advantages with respect to the traditional optimization techniques such as the following:

a) HS algorithm imposes fewer mathematical requirements and does not require initial value settings of the decision variables.

b) As the HS algorithm uses stochastic random searches, derivative information is also unnecessary.

c) The HS algorithm generates a new vector, after considering all of the existing vectors, whereas the genetic algorithm (GA) only considers the two parent vectors.

These features of HS algorithm improve flexibility and made it indispensable for optimization problems and produce better solutions.

To solve actual problems in an industry setting, a software engineering or a team of engineers must incorporate a development strategy that encompasses the process, methods and tools layers and the generic phases. This strategy is often referred to as a process model or software engineering paradigm. A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used and the controls and deliverables that are required. Several process models anguished on the development paradigms and can only had successfully delivered legacy systems, which are less compatible and have a monolithic designs. This is overcome by means of flexible design mechanism, using design patterns. Using design patterns a problem can be designed very easily within a short time to cover all aspects.

To explain in detail the Harmony Search, first the improvisation process of a skilled musician is idealized. When musician is improvising, has three possible choices: (1) play any famous piece of music (a series of pitches in harmony) exactly from musician's memory; (2) play something similar to a known piece (thus adjusting the pitch slightly); or (3) compose new or random notes. Zong Woo Geem et al. formalized these three options into quantitative optimization process in 2001, and the three corresponding components become: usage of harmony memory, pitch adjusting, and randomization [1]. As similar to the (GA) genetic algorithms the choice of the best-fit, the usage of harmony memory is importantly noticed. That ensures, the best harmonies will be carried over to the new harmony memory. In order to use this memory more effectively, it is typically assigned as a parameter $r_{accept} \in [0,1]$, called harmony memory accepting or considering rate. If this rate is too low, only few best harmonies are selected and it may converge too slowly. If this rate is extremely high (near 1), almost all the harmonies are used in the harmony memory, then other harmonies are not explored well, leading to potentially wrong solutions. Therefore, typically, we use $r_{accept} = 0.7 \sim 0.95$.

The second component is the pitch adjustment determined by a pitch bandwidth b_{range} and a pitch adjusting rate r_{pa}. Though in music, pitch adjustment means to change the frequencies, it corresponds to generate a slightly different solution in the Harmony Search algorithm [1]. In theory, the pitch can be adjusted linearly or nonlinearly, but in practice, linear adjustment is used. So we have

$$x_{new}= x_{old} + b_{range} * \varepsilon \qquad (1)$$

where x_{old} is the existing pitch or solution from the harmony memory, and x_{new} is the new pitch after the pitch adjusting action. This essentially produces a new solution around the existing quality solution by varying the pitch slightly by a small random amount [1,2]. Here *e* is a random number generator in the range of [-1,1]. Pitch adjustment is similar to the mutation operator in genetic algorithms. We can assign a pitch-adjusting rate (r_{pa}) to control the degree of the adjustment. A low pitch adjusting rate with a narrow bandwidth can slow down the convergence of HS because the limitation in the exploration of only a small subspace of the whole search space. On the other hand, a very high pitch-adjusting rate with a wide bandwidth may cause the solution to scatter around some potential optima as in a random search. Thus, we usually use r_{pa}=0.1 ~0.5 in most applications.

```
Harmony Search
begin
    Objective function $f(x)$, $x=(x_1,x_2, \ldots,x_d)T$
    Generate initial harmonics (real number arrays)
    Define pitch adjusting rate ($r_{pa}$), pitch limits and bandwidth
    Define harmony memory accepting rate ($r_{accept}$)
    while ( t<Max number of iterations )
```

```
        Generate new harmonics by accepting best harmonics
        Adjust pitch to get new harmonics (solutions)
        if (rand>$r_{accept}$), choose an existing harmonic randomly
        else if (rand>$r_{pa}$), adjust the pitch randomly within limits
        else generate new harmonics via randomization
        end if
        Accept the new harmonics (solutions) if better
end while
Find the current best solutions
end
```

Algorithm 1: Pseudo code of the Harmony Search algorithm.

The third component is the randomization, which is to increase the diversity of the solutions. Although adjusting pitch has a similar role, but it is limited to certain local pitch adjustment and thus corresponds to a local search. The use of randomization can drive the system further to explore various diverse solutions so as to find the global optimality.

The three components in harmony search can be summarized as the pseudo code shown in Algorithm. 1. In this pseudo code, we can see that the probability of randomization is

$$P_{random}{}^{=1-r} accept, \qquad (2)$$

and the actual probability of adjusting pitches is

$$P_{pitch} = r_{accept} * r_{pa}. \qquad (3)$$

V. HARMONY SEARCH FOR SOFTWARE ARCHITECTURE

Software Architecture problems that include reusing the components that are already implemented for applications, reuse of the design of one application into another, where there is a library of generic design patterns needs meticulous decision making which challenges about fitment and accuracy. Such problems, when they are solved, should project the finalized results for the developer to attain product development as in a final state. Using conventional selection methods, it is very difficult to ascertain the selection is final and best suited, for which that includes working out on so many parametric qualities. The best suited design is a dimensional pattern where the design pattern or software component is generically designed and be used for generalized application development. Thus, the selected stake and the inputs for selection are multi parametric; so a problem of multi-variable, multi-objective optimization persists.

Harmony Search is a strategic, meta-heuristic and algorithm-based-approach for engineering optimization problems with continuous design variables, especially they are much suitable for such software design problems. Quality and optimality of resulting design patterns or components is assured greatly with the application of Harmony Search strategic algorithm.

A. Implementation of Harmony Search for selection of a Design Pattern

Separating a software system into concerns is one way to deal with the increasing complexity of constructing large systems. However, not all concerns can e easily be modularized. Some concerns crosscut others. A crosscutting concern is one that is scattered throughout a system and is tangled with other core application concerns. A pattern story describes the application of patterns to a specific design.

Harmony Search Algorithm for Selection of Optimal Parametric Weights for deciding the design pattern.

The steps in the procedure of classical harmony search algorithm are as follows:

Step 1: Initialize the problem and algorithm parameters. The optimization problem is specified as follows:

$$\textit{Minimize } f(x) \text{ such that } x_i \in X_i,\ i = 1, 2, \cdots, N,$$

where $f(x)$ is an objective function; x is the set of each decision variable x_i; N is the number of decision variables, X_i is the set of the possible range of values for each decision variable, $X_i : x^L_i \le X_i \le x^U i$. The HS algorithm parameters are also specified in this step. These are the Harmony Memory Size (HMS), or the number of solution vectors in the harmony memory; Harmony Memory Considering Rate (HMCR); Pitch Adjusting Rate (PAR); and the number of improvisations (Tmax), or stopping criterion.

Step 2: Initialize the harmony memory. The HM matrix is filled with as many randomly generated solution vectors in possible range of decision variables, such as:

$$HM(j,:) \triangleq x^j = x^L + rand() \times (x^U - x^L),\ j = 1,2,...,HMS$$

Step 3: Improvise a new harmony. Generating a new harmony is called 'improvisation'. A new harmony vector, $x' = (x'_1, x'_2, \cdots, x'_N)$, is generated based on three rules: (1) Memory consideration, (2) Pitch adjustment and (3) Random selection. The procedure works as follows:

```
For each $\boldsymbol{i} \in [1,2, \ldots, N]$ do
        If rand() <> HMCR
                $x'_i = x_i^j \, (j = 1,2,...,HMS)$                // memory consideration
                If rand() <> PAR
                        $x'_i = x'_i \pm r \times bw$
                        $x'_i = \min[\max(x'_i, x_i^L), x_i^U]$          // pitch adjustment &
                                                                      truncation processing
                End
        Else
                $x'_i = x_i^L + rand() \times (x_i^U - x_i^L)$        // random selection
        End
End
```

Step 4: Update harmony memory. If the new harmony vector, $x' = (x'_1, x'_2, \cdots, x'_N)$ is better than the worst harmony in the HM, judged in terms of the objective function value, the new harmony is included in the HM and the existing worst harmony is excluded from the HM.

Step 5: Check stopping criterion. If the stopping criterion (maximum number of improvisations) is satisfied, computation is terminated. Otherwise, Steps 3 and 4 are repeated.

B. *The harmony search is applied into the software problem:*

Decision Variables in the HS Optimization are the Design Patterns what are to be chosen. Value Range of HSO algorithm is the volume/level of application of DP in the design. Solution Vector of HSO algorithm is the boundaries of applications of a DP in the design. Objective Function of HSO algorithm is the fine aesthetics of the DP in the current problem of design. Iterations of HSO algorithm is the practice on which the DP will adapt more suitably to the application. Memory Matrix of HSO algorithm represents experiences of applications of DP in the software development scenarios.

When the HS algorithm is implemented in Java (courtesy to Mohammed Fesanghary) the code reinstates the variables as follows: The key requirements of HS algorithm are; Harmony Memory Size (HMS), which includes the entropy of weights of the characteristics of the classes in the design pattern, Harmony Memory Considreation Rate (HMCR) is the rate where HS picks one value randomly from the musician's memory (HMS) 0 <= HMCR <= 1. 1- HMCR is the rate where HS picks one value randomly from total value range. The entropy of weights of each characteristic is prepared in a table for all the 23 GoF Patterns. Pitch Adjustment Ratio (PAR) is the rate where HS tweaks the value which was originally picked from memory, 1-PAR is the rate where HS keeps the original value obtained from memory. The selected group or range of weights from the weights of the entropy of characteristics is used to tune the selection of the pattern according to the size of the problem narrated in the pattern story. Number of Iterations (NI), is that the algorithm is iterated on the values to obtain the Local Optima of the weights of the characteristics and further the Global Optima of the desired design pattern. Fret Width is arbitrary length for the continuous variable only, which is also called (BW) band width. Fret is where the ridges on a stringed musical instrument to allow the tunes of the instrument to play. Pitch is the rate of vibrations in a tune. The Band Width determines the volume of the application of the design pattern to be applied for the problem. Finally the cadence is attained (cadenza) by the algorithm making a suggestion of the more suitability of the design pattern for the problem narrated in the pattern story.

I. EXPERIMENTAL RESULTS

The classical Harmony Search algorithm has been carried out in Java, with the specified parameters. The weights of the design pattern are predetermined for the known characteristics. The pattern Story is also formalized and weights for the component classes were built. The weights of the component classes in the pattern story become the question for identifying the desired design pattern. These weights are the used to tune the BW and PAR of the HS algorithm and it is implemented iteratively. Many observations are made on the output of the HS algorithm and the mean of the output weight is used to determine the weights of the design pattern. Assigning/designating the weights to the variables using Likert scale method is performed on the eights of the pattern story. We have experimented four Pattern Stories of a Text Editor / Word Processor and came out the compatible design patterns of Decorator, Adapter and Façade patterns.

```
   4.323968887329102       1.1349130868911743       1.0410350561141968       8.640802383422852
2.8220672607421875       2.9693503379821777       5.849714279174805       12.500911712646484
171.13710021972656  17.96278667449951

   4.104762077331543       1.7971773147583008       1.971657633781433       4.510588645935059
3.0633492469787598       3.7091116905212402       7.056185722351074       19.887590408325195
274.80279541015625  15.447534918785095

   3.8744843006134033       1.3624505996704102       1.3432064056396484       6.144593715667725
2.3172690868377686  3.457193374633789  5.991699695587158  17.424198150634766  264.8871765136719
15.042004108428955

   4.925594806671143       1.973921298980713       1.4136511087417603       7.418281078338623
3.0327324867248535       2.208566188812256       8.379963874816895       15.341063499450684
153.55018615722656  18.764180779457092

   4.733988285064697       1.4698853492736816       1.9643923044204712       6.246506690979004
2.665968894958496  3.162504196166992  7.11160945892334  23.451099395751953  411.63616943359375
17.08074152469635

   Execution time : 0.733 seconds

   best :10.026798338763362

    3.0009925365448 1.0026499032974243 1.000622272491455 3.0173819065093994 2.0051517199202826
2.2921719905926183 6.134219301562537 8.345452308654785 249.5384079426624
```

The observation drawn from the example of using weights of the decorator pattern, where the data shows the quantum of the limits of the properties to be used in the design pattern decorator for the application.

II. Conclusions

Design patterns are a technique for documenting solutions to recurring design problems, calculating the weights of the characteristics of the patterns and determined them as the desired ones for solving the problem is a challenging task. Various opinions come into light for which the design patterns are applied for the size of the problem and the domain. Formalization mechanism for the design patterns may be improved to an API level to directly know their characteristics and easy determination. The existing software tools which have done tremendous research made the design patterns complicated by applying constraints such as language, application thought and other perspectives. Unlike, the GoF patterns can be understood in plain to an API level and can be drawn into the programming environments.

References

[1] Zong Woo Geem, Music-Inspired Harmony Search Algorithm, © 2009 Springer-Verlag Berlin Heidelberg. ISBN 978-3-642-00184-0.

[2] Parikshit Yadav, Rajesh Kumar, S.K. Panda, C.S. Chang, An Intelligent Tuned Harmony Search algorithm for optimization, Information Sciences 196 (2012) 47–72, © 2012 Elsevier. doi:10.1016/j.ins.2011.12.035.

[3] Dennis Weyland, A Rigorous Analysis of the Harmony Search Algorithm - How the Research Community can be misled by a novel Methodology, International Journal of Applied Metaheuristic Computing, volume 1-2, April-June 2010, pages 50-60.

[4] Omran, M. and Mahdavi, M. (2008). Global-best harmony search. Applied Mathematics and Computation, 198(2):643-656.

[5] Uwe Zdun and Paris Avgeriou, A catalog of architectural primitives for modeling architectural patterns, Information and Software Technology 50 (2008) 1003–1034, © 2008 Elsevier.

[6] Donald Metzler and Hugo Zaragoza, Semi-Parametric and Non-parametric Term Weighting for Information Retrieval, Yahoo! Research, © 2010 Yahoo Inc.

[7] G Antoniol, G.Casazza, M. Di Penta, R. Fuitem, Object Oriented Design Pattern Recovery, The Journal of Systems Software, Vol 59. 2001, 181-1196. © 2001 Elsevier.

[8] Frank Buschmann, Pattern Oriented Software Architectures, © 2001 Reprint, John Wiley & Sons, Germany.

[9] Christopher G. Lasater, Design Patterns, © 2007, Wordware Publishing, Inc.

[10] Dan Lockton, David Harrison, Neville A. Stanton, Design with Intent, Equifine Windsor, Berkshire, UK, © 2010.

[11] Chris Ford, Ido Gileadi, Sanjiv Purba, Mike Moerman, Patterns for Performance and Operability: Building and Testing Enterprise Software, Auerbach Group, © 2008 by Taylor & Francis Group, LLC.

[12] Richard P. Gabriel, Patterns of Software: Tales from the Software Community, New York, Oxford, © 1996, Oxford University Press.

AUTHORS PROFILE

Dr. A. Rama Mohan Reddy is currently working as Professor, Department of CSE, SVU College of Engineering, S V University, Tirupati, A.P. He has completed his B.Tech from JNT University, Anantapur and M.Tech from NIT, Warangal. He has received his Ph.D. from SV University in Software Architecture. His other areas of interests include Data Mining and Object Oriented Analysis & Design.

A.V. Sriharsha is B.Tech in Computer Science & Engineering from Andhra University and M.Tech in Information Technology from Sathyabhama University, Chennai. He is pursuing PhD under the esteemed guidance of Prof. A.Rama Mohan Reddy, Department of CSE, SVU College of Engineering (Autonomous), S.V.University, Tirupati, A.P. His main research interest includes Data Mining, Information Retrieval, Database Management Systems and Knowledge-based Software Architectures.