Knowledge Maps = Stable Pattern Languages

M.E. Fayad, PhD and H.A. Sanchez
mefayad@gmail.com, m.fayad@sjsu.edu,
huascar.sanchez@pearson.com, huascar_sanchez@comcast.net

Abstract:

Pattern languages form the strong groundwork for any discipline understanding. Its uses have spilled over to the software engineering field, to precisely describe past experiences and better understand software architectures conceptualization and realization, along with, how their building blocks are insightfully woven seamlessly to satisfy a determined purpose resolution. Current representations of pattern languages [5, 6, 7] strives hard in providing the sufficient means or guidelines to build software architecture out of patterns; bringing as a consequent problems, such as poor traceability [3], lack of stability, poor adaptability, out of context understanding realization, are more prominent within the pattern language implementation. To overcome these problems in traditional pattern languages, we have provided both a set of quality factors to evaluate pattern language definition, accuracy, and application; and also a newly enhanced Pattern Language representation, driven by Software Stability Concepts [8, 9, 10], called Knowledge Map, which provides many significant benefits to the software architecture’s development practices.

1. Introduction:

The concept of Pattern Language is crossing over to the software engineering field to precisely describe, prior experiences and the processes that stem from them, in a cohesive and clear, not less important, language where practitioners can create their own environment. It has emerged as a non-formal language, that encompasses a set of related patterns that form the basis for any discipline learning and understanding [1]. Each of the encompassed pattern per se is twofold, the workhorse to handle recursive problems in a determined context of applicability, and an insightful way to understand and learn an individual concern in a determined context. Therefore, when these patterns are skillfully woven into a cohesive whole, practitioners will be able to combine them in any way they want to satisfy a determined purpose resolution [11].

Product of the proliferation of pattern languages in areas as diverse as biology, and education, raises many questions such as “what defines good pattern languages?”, “What are the pervasive criteria that must be satisfied by all pattern languages in different contexts?”, and “What is the behind the scenes language that guides the sewing of patterns together?”. These questions are not fully addressed in the current pattern languages representations [5, 6, 7]. Therefore, problems such as poor traceability [3], lack of stability, poor adaptability, out of context understanding realization, etc. in currently existing solutions, become more prominent when used by software practitioners.

In order to answers those questions and overcome arisen problems, we hereby provide a set of quality factors to evaluate pattern language definition, accuracy, and application; and also a new enhanced Pattern Language representation, called Knowledge Map, which provide significant benefits to the software architectures development practices. These benefits are described further in the essay. To assess our paper contributions, we will survey a vast number of examples, within the existing genre of Pattern Languages, and analyses found many commonalities in these forms, with special reference to the instances surveyed. The resulting pattern languages would be compared and contrasted by using already defined quality factors against the pattern languages that are driven by software stability concepts.
(Knowledge Map). For simplicity’s sake, we will briefly touch these benefit’s descriptions, nonetheless we make sure that these descriptions are clear for the reader.

Throughout our survey, each of the existing patterns and pattern languages, along with their counterparts (stable patterns, knowledge map proposed in this paper) would be validated against several practical scenarios, that require their use and quality factors. The main purpose of this experiment is both to identify, whether selected patterns completely satisfy the needs expected in a determined scenario or not; and to surfaced the superiority of this new generation of pattern languages called Knowledge Maps. Due to pages limitation, the actual step by step comparative study between types of pattern languages will be omitted in this essay, since it is a big and wide topic that is worthy of a new essay. We will mainly concentrate on the results of the study instead.

1.2. Motivation and Contributions:

The motivation of our work is to provide a stable pattern language, that covers the realization of stable software architectures that are capable of evolving through time, without the concern of possible collapses and unsatisfactory requirements fulfillment. Our main goal is to provide a stable pattern language or Knowledge map, along with some of its significant properties, that generates a myriad of software architectures, within context, scalable, and adaptable to transient customer needs. This new pattern language provides a strong groundwork for an insightful classification of the ultimate concerns governing a particular discipline.

In this essay, software stability concepts approach is introduced as the key engine driven the materialization of our Knowledge map or topology of patterns. This knowledge map will come to enhance pattern language representation and realization in different disciplines, by assuring both their proper use, when describing a particular domain, and satisfactory concern classification. We will also show the structure of our knowledge map further down in the essay.

1.3. Essay Organization

This essay is stratified as follows; Section 2 describes the background for our Knowledge map, and illustrates our contributions to software community, such as the quality factors for pattern languages comparison, and the knowledge map structure, its benefits and main properties. Section 3 concentrates on the results from the comparative study, between current pattern languages structure and our new stable pattern language using the aforementioned quality factors. Our last remarks are stated in Section 4, while Section 5 provides a list of references.

2. Knowledge Map Background/Structure

In this essay, we name any pattern languages representation and approaches not driven by software stability, as traditional pattern language approaches, and the ones driven by software stability concepts as Knowledge Map. To better understand this “software stability concepts driven” pattern language, we will fully describe it in Sub-section 2.2. Before any comparison between these two pattern language representations and generated outcomes, we have enlisted in a significant endeavor, where we extracted the essential aspects that any pattern language must include within their crafting, along with the main properties product of these essential aspects.

These essential aspects are referred to as Pattern Languages’ quality factors. These quality factors are described in Sub-section 2.1.

2.1 Pattern Languages’ Quality Factors
Defining a good and robust pattern language is a major challenge, especially when dealing with heterogeneous domains’ characteristics and behaviors, and when there is not a complete and systematic definition process to ably support it. This sub-section introduces certain quality factors for pattern languages definition and application. However, for the sake of simplicity, we are not intending to provide a complete study of pattern language quality factors; instead, we will briefly provide their main idea, and to show, whether they are addressed in current pattern languages definition and application or in our proposed knowledge map.

It is worth to repeat that the language of a group of patterns forms the groundwork for any discipline [1]. In other words, this language could be seen as a mirror of a discipline collaborative experience gained over generations, which was placed on writings for future knowledge sharing. Therefore, their great proliferation across different domain was always expected; nevertheless it requires certain criteria for their proper definition and utilization:

1. **Stable Nature**: We mean by stable nature, as the capacity of pattern languages to be carefree to the forces of changed, by detaching their stable core knowledge (endured knowledge) from unstable or specific pieces of knowledge bound to specific contexts. Nevertheless, to provide a space when these two elements can interact and form context-specific knowledge.

2. **Full Scalability**: This concentrates on the ability of patterns languages’ knowledge scope, along with their enclose elements, to adapt to evolving needs and insights without unnecessary effort [2], by means of either adding/removing patterns to/from the pattern language without any arise complication or collapse, or extending/reducing their peripheral knowledge scope [2].

3. **Rationale Traceability**: This quality mainly concentrates in the ability of pattern languages’ enclosed patterns or elements, along with their actual implementation, which are part of a selected path, to be successfully traced back to their original goal or usage rationale [4].

4. **Generative Nature**: This quality represents both the ability of pattern languages to reproduce or originate complete solutions, based upon a set of intertwined patterns, and the patterns connective rationale: patterns interconnection and patterns selection flow guidance.

5. **Adequacy**: It is the ability of pattern languages of satisfying the requirements of the intended and established purposes, along with their consequences. In other words, the pattern language goodness of fit. This quality extends from:
   a. visualizing and monitoring each of the elements in the pattern language,
   b. efficient modeling and documentation means,
   c. the ability of attaining and representing objective knowledge and the rationale of what is known already,
   d. to searching and recognition capabilities.

6. **Verification and Validation ability**: This quality concentrates in providing the means for verification and validation of selected paths in the pattern language and decides whether the selected path is the most suitable selection or not.

These quality factors would be the means for comparing traditional patterns language’s work with the one proposed in this essay (Knowledge map). Next sub-section will provide an exhaustive description of the knowledge map structure, mantra, and application.

### 2.2 Knowledge Map Structure and Rationale

A knowledge map is a topology of patterns that is driven by the principles of software stability concepts [8, 9, 10]. In this sub-section, we provide its structure, mantra, and the rationale-driven language use to discover and visualize elemental pieces of knowledge (patterns), how to organize them, and how to relate them to formulate an accurate solution in contexts, that shares the same core knowledge (rationale or goals, and capabilities).
Building a knowledge map for a determined discipline involves myriad skills, knowledge and steps beyond the identification of the tangible artifacts that are bound to a specific context of applicability. It also requires systematic capture and fully understanding of the domain, in where our solution would be laid down and expanded. That includes, describing the problem not from its tangible side, but focusing more on its conceptual side; describing underlying affairs with respect to the problem, and the elements required to fulfill them.

The following illustrates how the knowledge map will look like. Its global representation is tied up to the significant mantra “Divide and Conquer”. This mantra is realized in the 4 different points of the knowledge map. For instance, at a global level, this knowledge map is stratified into 4 concerns: Analysis concerns called Goals, Design concerns called Capabilities, Development concerns called Development scenarios, and Deployment concerns called by the same name. In addition to this, each of the pieces of knowledge enclosed within the knowledge map’s boundary (patterns) is expanded into software stability concepts’ three layers of concerns, which are Enduring Business Themes (EBTs), Business Object (BOs), and Industrial Objects (IOs). For a better picture of this, please refer to figure 1.

![Knowledge Map structure](image)

**Figure 1.** Knowledge Map structure

As you can see, the mantra “divide and conquer” is applied throughout the Knowledge map’s structure.

2.2.1. **Knowledge Map Key Properties:**

Our stable pattern language structure or Knowledge Map has the following properties:
1. Knowledge Map Partitioning: This means that Knowledge Maps partition a particular domain by twofold. The first level of partition refers to the Concerns classification partition, where concerns are allocated based on their rationale: Domain Goals, Goals’ Capabilities, Development Scenarios, and Deployment scenarios. The second level of partition refers to individual concern’s partitioning, where each concern structure is divided into two or more layers: EBTs, and BOs, or EBTs, BOs, and IOs. This particular property of Knowledge Maps makes a strong reference to a significant object oriented technology’s mantra, which is "divide and conquer."

2. Knowledge Maps Intersection: Through the sharing of determined concerns among two or more Knowledge Maps and the Knowledge map proper partitioning, the Knowledge map is able to enhance the associated domain understanding, its extension, and identifiable composition, while at the same time allows great levels of traceability across its entire structure. The following image (figure 2) shows the intersection of two knowledge maps, this extends the level of understanding of software practitioners with respect to intersecting knowledge maps.

3. ROI: Since the driving forces of Knowledge maps are "stability and reusability", via the use of stable patterns, or insightful and stable solutions for a set of recurrent problems, a vast number of software architectures will be cheaper to construct, and with a significant reduction in cost and innovation, which means a faster and quicker ROI.

4. Infinite Number of stable architectures: Exploitation of useful patterns synergies from available stable patterns to generate a vast number of stable software architectures, accelerating software architecture production and workflow pattern selection, while at the same time providing an intuitive process execution to address reusability, scalability, adaptability, etc.

5. Out of Scope Assessment: Domain understanding and assessment sit side by side, in the conceptualization of the knowledge map, which can guarantee the definition of within context concerns across the knowledge map structure. This will also guarantee a unified and accurate patterns topology, spanning all the building blocks of a particular domain or discipline.

The rest of the essay concentrates in describing the systematic approach to implement this knowledge map or stable pattern language.
2.2.1. Knowledge Map behind the scenes methodology:

Knowledge Map conception follows a goal driven methodology. This goal-driven methodology covers the entire software development cycle, from the requirements gathering phase (Analysis) to the actual implementation, applying verification and validation concurrently at each of the phases involved in the software life cycle. The heart of this goal-driven methodology is the requirement elicitation phase, in where Analysts, Designers, etc. discover the goals, and purposes of the system under construction. The discovery of these goals and their knowledge realization will orient in the resolution or coin of the rest of phases‘ outcomes. In the case of requirements elicitation, we are talking about Stable Analysis Patterns; for designing phase, and knowledge decomposition, we are talking about Stable Design Patterns and Stable Architectural patterns respectively. This is illustrated in figure 3.

With respect to transient aspect bound to specific contexts, the goal-driven methodology allows the integration of these aspects to the knowledge map, by means on utilizing extension points called hooks [2]. These hooks serve as a perfect gluing mechanism, that allows adding volatile context specific aspects to the endured infrastructure formed by EBT and respective BOs. This does not sound earth-shattering; in order to better picture this, we provide a different view of figure 2 including the hooks and some IOs (Figure 4). These IOs are artifacts, that change according to specific needs and contexts.
The process of defining Knowledge map involves four main steps, which can be perceived as the basis for our pattern categorization: 1. Goals or Classification, 2. Capabilities/Properties, 3. Development or Scenario Development, and finally 4. Discipline Deployment across multiple domains. The outcome of these main steps will be a set of interrelated patterns, that interact together to serve a particular purpose.

When putting these patterns to work together in a synergetic manner, the actual interaction will create the illusion of narrating a story tell, where the topics or goals selection will lead to a more precise capabilities choosing and a more efficient problem resolution. After selecting these goals, the way and flow of how these capabilities will be combined and followed are quite infinite. As a result of this infinite intertwining, there will different ways to resolve a particular problem.

The actual methodology covers the following steps:

1. Goals or Classification:

   This step is concerned with surfacing the implicit goals hidden within a particular discipline core knowledge. This process requires the capture and fully understanding of the context, where a solution would be laid down. That includes, describing the goals not from its tangible side, but focusing more on its conceptual side.

2. Capabilities or Properties

   The second step concentrates in the discovery on those recipes that are required to fulfill the stated goals and purposes of the discipline under discourse. Without those concepts or stable patterns, there will be a vague understanding (almost none) on how these goals will be achieved.

3. Development Scenarios:

   The third step is concerned with: 1- the myriad of development scenarios, in where a discipline can be developed. These development scenarios are realized through the distinct routes or paths taken due to the interactions among the involved patterns. The product of this association is known as stable Architectural Patterns. Each one the complete routes taken will represent a distinct application domain or context in, where a particular discipline would be used. And 2 – how the discipline
particular’s language would be implemented across dissimilar domains based upon the utilization of transient artifacts.

4. Deployment:

The last step, as its name is already stated, deals not only with how a determined discipline’s knowledge would be deployed across different application domains, but also with the representation of the artifacts or patterns, that will aid the deployment process.

The rationale of discovering and stratifying our knowledge map by means of using a systematic approach, which involves four main steps, and ends up with four categories, is to mainly facilitate the findings, execution order, and description of the stable patterns embodying the concepts or building blocks of any discipline’s domain. Next section will provide the results of a comparative study between traditional pattern languages approaches and our stable one (Knowledge map).

3. Pattern Languages - A Comparative Study Results:

In this section, we provide the results from the comparative study performed over both traditional pattern languages and the propose knowledge map. These outcomes will be described in a step by step manner, which is driven by Pattern Languages quality factors and properties.

3.1. Traditional Pattern Language Approaches:

These are some of generated outcomes from the perusal of traditional pattern languages.

1. Missing indicators and guidelines for assuring a “within-context” behavior for the pattern language’ provided patterns. There is not a magic recipe for deciding which patterns to use and add in a particular pattern language. Therefore, practitioners are infected with a “keep adding what it seems to satisfy my requirements” inaccurate behavior. These actions at the end will provide more problems than visible benefits for software practitioners, when using traditional pattern languages.

2. No identifiable classification of patterns’ rationale within the pattern language structure, along with the lack of concerns partitioning, which facilitates an efficient distribution of pattern responsibilities. This will bring many serious problems, like macho class problem, where all responsibilities are positioned in one class, or high coupling and low cohesion problems, etc.

3. Lost of traceability across all the different stages of software development, especially when tackling the deeper levels of pattern implementation. There is a traceability problem in the instantiation processes of traditional pattern language, which result in instances, that cannot be traced back to their original design patterns. The instances were deduced from the design pattern, but the internal structure of design patterns is invisible in traditional pattern languages [3].

4. No clear and systematic manner to compose patterns together and create software architectures. This is performed on ad-hoc basis, therefore most of the resulted solutions accuracy is highly questionable, and open to serious contamination and ripple effects [2].

5. Pattern Language’s pattern lost of generality, since most the patterns added to the pattern language are out of context, and usually their implementation instances are lost within the pattern language implementation. There is not way to pinpoint the pattern instances, once the entire pattern language has been implemented [3, 4].

3.2. Stable Pattern Languages – Knowledge Map:
Defining stable pattern languages is a new movement, therefore there are few pattern languages that were defined under the wings of software stability concepts principles, and use in our comparative study. The following represents how the Knowledge map has overcome traditional pattern languages problems.

1. Systematic capture processes and full understanding of the domain, where a targeted solution must exist. This will provide a Unified domain knowledge discovery and understanding approach, spanning all different categories for domain-neutral patterns.
2. Great Reuse and on demand context adaptation, via an identifiable stable core and its extension points called Hooks, i.e. Type Patterns. Transient aspects, then are attached to the core according to context specific needs. These qualities will greatly reduce cost of adaptation or upgrades.
3. Full traceability capability across stable pattern language implementation. This is accomplished by describing a two-way mapping relationship between stable design patterns and their implementation instances. Therefore, traceability, maintainability, and stability become visible assess for your software architectures.
4. Knowledge maps provide unprecedented flexibility, by enabling the business to add or remove functionality from their system on a real-time basis. Therefore, there will be a Rapid ROI, as software architectures are cheaper and faster to construct and reuse.
5. Rapid software architecture definition and application, through a systematic workflow pattern process, and identifiable pattern composition process, which, as a whole greatly satisfy the features and functionality that are needed by the customer throughout the lifecycle of a business.

4. Last Remarks:

The objectives of this essay were to introduce a new enhanced and stable pattern language, known as Knowledge Map or topology of stable patterns, and also show its superiority over traditional pattern languages. We have achieved these objectives by applying Software Stability Concepts and its SSM to the knowledge map analysis, conceptualization and elements’ discovery process, and also by enlisting in a comparative study against its counterpart traditional pattern language representations. Thereby, we were able to determine a superior representation of a stable and enduring structure for Knowledge Maps, that encloses any discipline perusal and definition efficiently, making sure that their elements are classified according to these elements rationale. The essay also provides a significant description of the Knowledge Map structure, and its properties along with the benefits, challenges and constraints.

5. References:

