ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 8(24), DOI: 10.17485/ijst/2015/v8i24/81850, September 2015* ISSN (Online) : 0974-5645

Detecting Resemblances in Anti-pattern Ideologies using Social Networks

Saini Jacob Soman*

Department of CSE, SNG College of Engineering, Kadayiruppu - 682311, Kerala, India; sainijacobs@gmail.com

Abstract

A key argument for modelling knowledge in ideologies is the simple reuse of the facts. However, nearby reliability checking, current ideology engineering tools give only essential functionalities for analyzing ideologies. Since ideologies can be considered as graphs, graph analysis techniques are an apt answer for this necessity. The anti-pattern ideology has been recently proposed as a knowledge base for SPARSE, an intelligent system that can detect the anti-patterns that exist in a software project. However, apart from the excess of anti-patterns that are intrinsically informal and vague, the data used in the anti-pattern ideology itself is many times inexactly defined. We exemplify in this paper the benefits of applying social networks to ontologies and the Semantic Web and discuss which research themes happen on the edge between the two particular fields. Particularly, we confer how different ideas of centrality portray the core content and structure of ontology.

Keywords: Anti-patterns, Bad Code Smell, Sparse

1. Overview

Anti-patterns symbolize the latest concept in a series of radical changes in computer science and software engineering thinking. As we move towards the 50-year mark in developing programmable digital systems, the software industry has yet to determine some basic problems in how humans interpret business concepts into software applications.

An assessment of hundreds of corporate software development projects showed that eight out of ten software projects are considered ineffective. About one third of software projects are cancelled. The remaining projects delivered software that was normally double the expected budget and obtained twice as long to build up as initially planned[9]. These repeated failures is highly valuable, however in that they offer us with practical knowledge of what does not work and during study: why. Such study, in the dialect of Design Patterns can be classified as the study of Anti-patterns.

Design Patterns present the most successful form of software control yet available, and the whole patterns progress has gone a better way in codifying a brief terminology for transmitting complicated computer science thinking. Anti-Patterns are a natural addition to design patterns, concentrating on the broad and ever-growing assortment of repeated software problems in an effort to understand, avoid and recuperate from them. Anti-Patterns are new tools that bridge the gap between architectural theories and real-world executions[7].

The design of Anti-patterns is that there are various fruitless behavior modes in software engineering which result in similar problems[9], and can be treated in various and possible ways.

2. The Problem

Applying an antecedent prospering resolution to the same drawback is one amongst the established ways that during which humans solve issues. This approach is thought

*Author for correspondence

as nonliteral, allegoric, case-based reasoning and may be a documented machine learning technique employed in computer science systems. This approach in drawback finding needs expertise on previous issues and each success and failure should be experienced so as to collect the desired experience for future drawback finding. Moreover, whereas it should be fruitful to check prospering ways that developers solve issues mistreatment style patterns these mechanisms haven't been employed in project management as they will not accommodate such advanced descriptions.

The most common mistake created with style patterns, is that the application of a specific style pattern within the wrong context or setting. Antipatterns redefine the idea of style patterns in a very new type that makes an attempt to resolve this downside by providing careful templates[6] that state the causes, symptoms and consequences of an anti-pattern. Finding out failures and learning from mistakes could be a much more appropriate approach for project management. Antipatterns are the primary mechanisms that adopt a "negative solutions" perspective at software package development and are the primary to simply accept and handle the potential of a software package project for failure[7]. The subsequent parts during this section discuss the motivation behind the study of antipatterns, describe the contribution of the paper and at last present the organization of this paper.

The motivation behind why Software Project Anti-pattern Knowledge Management is the subject of this proposition is on account of it remains logically unexplored, as well as on the grounds that anti-patterns can encode and oversee venture administration, as well as any sort of programming advancement information that prompts reoccurring tricky practices or dangerous results. It makes considering the route with which managers and programming experts can adequately utilize anti-patterns an important scholastic wander. Helpful research in this bearing is significant in giving a specialized answer for the issues that as of now torment anti-pattern research. An alternate predominating reason is that few issues identified with anti-patterns have not been dissected.

Notwithstanding of the extensive number of accessible antipatterns, utilizing anti-patterns as a part of programming task administration remains tricky because of an arrangement of issues that distress the antipattern research. The primary issue that forbids the more extensive reception of anti-patterns is the absence of a regular vocabulary[4] of terms that might be utilized between individuals and programming equipment. The absence of formalization does not permit the configuration of models, architectures and programming frameworks that could profit the engineering of anti-patterns.

3. Anti-Pattern Models

By creating anti-pattern formalisms, the learning encoded in antipatterns could be formalized and handled by programming tools. At a learning representation level, the anti-pattern ideology can focus the classifications of things that exist in an anti-pattern and set the ontological duties of the task supervisor, framework planner or requisition architect[1]. Both of the recommended formalisms offer a medium for proficient reckoning on the grounds that they don't just speak to learning; additionally encode data in a structure that could be transformed productively by programming. Moreover both BBNs and ideology offer a medium for human representation[3] that might be utilized by project managers within request to have a regular vocabulary of terms with a strong scientific establishment.

The anti-pattern ideology encodes implied software project management information into a computer readable structure and permits the offering and reuse of this knowledge by programming tools. Moreover, the issue of catching and quantifying doubt in the antipattern ideology is tended to by including the ideas of anti-pattern BBN models and their corresponding OWL ontology in the outline of the non-specific antipattern ideology.

4. Related Literature

Expert systems have been extensively used in most dissimilar settings including teaching[12], medicine[7] and security decisive systems[4]. The World Wide Web contains plentiful examples of expert systems. However, expert systems are not a solution and can be wrong[11]. Adams[2] has presented some thoughts for the expert system design that want to be addressed when the system is used via the web. The author concludes that the viability of giving expert system capabilities over the web depends upon the exact situation for which the expert system is developed.

Four primary classes of antipatterns have been recognized in the writing:

- **Development Anti-patterns:** Describe specialized issues and results that are experienced by programmers.

- **Architectural Anti-patterns:** Describe normal issues in how frameworks are organized.
- **Environmental Anti-patterns:** Describe issues brought about by a predominating structure or social model, which are the aftereffect of uncontrolled socio-political strengths.
- **Managerial Anti-patterns:** Describe issues brought about by a project director or administration team in programming and development firms.

Software project management anti-patterns can deal with all parts of a software project more successfully by bringing understanding into the reasons, manifestations, results, and by giving great repeatable results[8].

Four group[11] formats is a more official layout that further incorporates different components, for example, the inspiration, members, related examples, known utilization and coordinated efforts.

In any case, it is impossible that each of the three events could be determined in the same way[5]. A more detailed report is the Mini-Antipattern template, which explains the two solutions of the anti-pattern, the difficult answer and the re-factored result[13].

4.1 Anti-pattern Interrelationship

Anti-patterns can seem separated yet can likewise be connected with different anti-patterns. The later sort is alluded to as cooperating anti-patterns[9] and is clear when a project management anti-pattern causes a software advancement anti-pattern or a construction modelling anti-pattern. It is vital to comprehend the extravagance of anti-pattern interrelationships so as to have the ability to determination anti-patterns exclusively as well as location them as an assembly of interrelated anti-patterns. This proposition keeps tabs on software project management anti-patterns however considers the way that these anti-patterns could be connected with different sorts of anti-patterns. Anti-patterns could be connected through the properties of following Table 1.

Table 1. Attributes relating to software project management anti-patterns

Causes	A list which recognizes the causes of this antipattern.
Symptoms	A list which comprises the observable symptoms of this antipattern.
Consequences	A list which comprises the penalties that result from this antipattern.

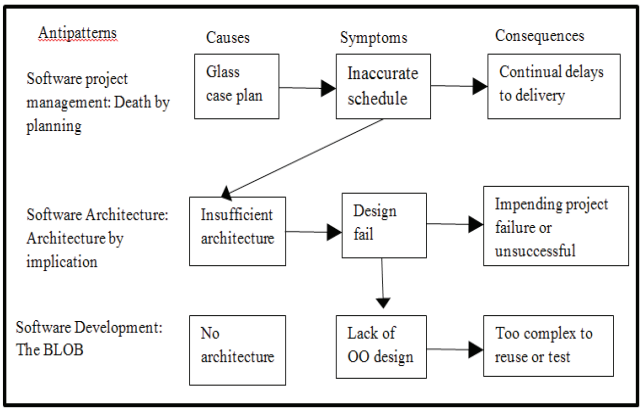

Figure 1. Dataset of anti-patterns relations through their attributes.

Figure 1 outlines a sample relationship between 3 separate anti-patterns, through their reasons, manifestations and outcomes. These are the primary qualities of an anti-pattern that characterize how an anti-pattern is connected with different anti-patterns.

5. Methodology

Here our methodology is that ideology based method to characterize and apply the properties of hostile to examples, bad code smells, refactoring (OABR), and the relations between them. We have gathered, sorted out and arranged the properties of the related ideas and we extended the properties for bad code smell by making a quality list used to prioritize bad code smells with the objective of giving backing to recognizing which bad code smells ought to be evacuated, or endured. We then made layouts dependent upon properties for extra bad code smells and refactoring examination. We likewise created scientific categorizations for against examples, refactoring and bad code smells to give progressive characterizations. Here additionally we have demonstrated the phrasings and relations spoke to by the essential Descriptive Logics (DL) used to characterize ideology dialect. We created an OABR base including hostile to examples, related software issues and location dependent upon the properties, taxonomy and non-taxonomy relations. At last, we portray making, gaining entrance to, saving, questioning and mapping of OABR with the ontological devices, stages and ideology registries/repositories.

5.1 Antipattern Properties

Numerous properties of existing anti-patterns were characterized by[2] and[10] independently. In this exploration, we chose properties, for example, name, causes, results, symptoms and refactoring. Different properties, for example, root causes, variations, background and general structures are not included as they are dependent on the developer's personal experiences. Additionally, an excess of properties for a particular idea will expand the multifaceted nature as per the essential standard that —the more expressive the language, the harder the thinking[11].

5.2 Refactoring Properties

We characterized the refactoring properties as name, situation, and mechanics. The name of a refactoring generally comprises of an operation and an object. Case in point, for the 'Remove Middle Man refactoring', Remove is an operation while - Middle Man is an item. The situation property gives portrayal to each one refactoring about when it will be connected. The mechanics property depicts how to apply methods step by step for every refactoring to tackle the related issue.

5.3 Bad Code Smell Properties

A goal of this research is to give a more formal and predictable documentation of properties to make every bad code smell less demanding to recognize and distinguish. Current terms of bad code smells are depicted and sorted out in a fairly casual and conflicting way. We analysed initial properties of bad code smells as name, symptoms, measurements and refactoring. Symptoms property depicts how to search a bad code smell.

Software Product Metrics measure programming tools at distinctive development stages, extending from measuring the complexity of programming design to the extent of the final source code. The measurability of a bad code smell relies on upon the size, the multifaceted nature and the structure of the bad code smell. Some bad code smells, for example, long Method might be effortlessly recognized by accepted programming tools such as Cyclomatic complexity and Halstead measures

5.4 Tools and Platforms

A software application permits research problems to be resolved by using software solutions. This level of research also provides the chance to check the formerly defined theory and methods for seizure and significance.

SPARSE is web-based collaborative ontology control software in which the anti-pattern ontology can be augmented. Web-Protege has been used in order to propose a Web-based interface that uses the Protege platform, in order to permit combined ontology editing as well as gloss and selection of both ontology mechanism and ontology changes.

5.5 Technologies for Communities

Ontology is an open system endorsing broad use and sharing. Its growth and justification depend on the contribution from the users of related community. The standard way is to list the ontology with an ontology search engine, or with a storehouse to make the ontology noticeable to the community. The feedbacks from the community will make the ontology more reliable and consistent.

6. Conclusion and Future Works

Against examples and bad code smells portray perpetual issues that influence programming quality. Refactoring can help fathom hostile to examples and bad code smells. In this examination, we created an ontological base, OABR, indicating the relations between against examples, bad code smells, and refactoring to help in the distinguishing proof and determination of their partnered issues.

The paper draws information from different sources to speak to, model and examine antipattern information with a specific end goal to resolution issues that encompass the innovation of antipatterns. The effects needed structure the work of the creator and his associates have gone far in creating hypothetical models, applying strategies and creating a product device with a specific end goal to aid the anti-pattern location process. Be that as it may, the work might likewise raise numerous other essential issues that need to be explored further. These issues must be dealt with and not so much in place of vitality.

The Dependency Structure Matrix (DSM) has been proposed as a strategy that pictures and breaks down the conditions between related qualities of software project management anti-patterns. The methodology was exemplified through a DSM of 50 traits of 25 related software project management anti-patterns that show up in the literature and the Web. A good set of blended anti-pattern

information including advancement, design and managerial antipatterns will uncover the appropriateness of the technique in such settings.

The progressing tests and future work of the exploration incorporate the accompanying:

- Obtain more inputs from the software group to grow OABR and set requirements for the class properties, given that the improvement of OABR is an iterative methodology.
- Develop OABR registries and related web administrations, making it less demanding for clients to recognize and test new bad code smells, hostile to examples, and refactoring.
- Expand or make another ideology by consolidating or selecting OABR with different ontologies about software improvement, for example, configuration designs, software measurements, and software quality attributes.

7. References

1. Basilevsky A. Statistical factor analysis and related methods: Theory and application. New York: Wiley; 1994.
2. Charles P. Pfleeger: Security in computing. 2nd edition. Prentice Hall PTR; 1996.
3. Degenne A, Forse M. Introducing social networks. Thousand Oaks, California: Sage Publications; 1999.
4. Gollmann D. Computer security. John Wiley and Son Ltd; 1999.
5. Gamma E, Helm R, Johnson RE. Design patterns. Elements of reusable object-oriented software. Addison-Wesley Longman; 1995.
6. Valiente G. Algorithms on trees and graphs, Springer; 2002.
7. Fowler M. Refactoring. Addison-Wesley; 1999.
8. Lanza M, Marinescu R. Object-oriented metrics in practice. Springer; 2006.
9. Fenton NE. Software metrics: A rigorous approach. Chapman and Hall Ltd; 1991.
10. Devanbu P. GENOA - A customizable, front-end retargetable source code analysis framework. ACM Transactions on Software Engineering and Methodology. 1999; 177–212.
11. Scott J. Social network analysis: A handbook. Second Edition. Thousand Oaks, California: Sage Publications; 2000.
12. Levenshtein VI. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady. 1988; 707–10.
13. Wasserman S, Faust K. Social network analysis. Methods and applications. Cambridge: Cambridge University Press; 1994.