Change Requests Artifacts to Assess Impact on Structural Design of SDLC Phases

Rudra Kumar Madapudi
Assoc. Professor,
Dept. Of CSE
AITS, Rajampet

A. Ananda Rao, PhD.
Professor & Principal
JNTU College of Engineering,
Anantapur

Gopichand Merugu
Associate.Professor
CSE Department, BVRIT,
Narasapur, AP, India

ABSTRACT

Current escalating demands on software, software developers to be obliged to generate software that can be altered, which escape from the risk of mortifying the software structural-design of the "SDLC phases". Degraded software structural-design is problematic because it makes the system more prone to defects and change requests turn to be costlier. The impacts of change requests to software can be hard to determine. One way to determine these consequences is to artifact the causes and effects caused by change request. A software change artifact allows to assess the effects of a change using different criteria such as causes to apply the change to be requested, change request type and the software module influenced by that changes. Once these artifacts identified then these artifacts can be used to scale the potential impact of the change. Another benefit of defining artifacts of the change-requests are that it allows engineers to develop a common approach to deal with changes that have similar in defined artifacts, rather than addressing each change individually. This paper introduces a mechanism that defines artifacts of the change-request to assist developers in measuring the impact of a software change on the structural-design of the SDLC-phases.

Keywords:

Artifact, change request, SDLC, software engineering, risk prediction

1. INTRODUCTION

To begin development, a set of requirements must be agreed upon by the developer and the customer. He stated that the software undergoes never-ending maintenance and development that is driven by the difference between its current capability and what is required by the ever-changing environment [1]. It is the foundation for the development of budgets, schedule, tests, and design [2]. Therefore, developers must have effective mechanisms to manage the change process [3]. It has been hypothesized that, likely increase in cost to handle the defects in the process of software development, a potential change in software requirements applied later in the life cycle will be more difficult and costly to implement [4]. The environment change could require changes in protocols and standards necessary for communication with other systems. Software requirement changes are common and frequent in different phases of the life SDLC. A change request should contain all the information necessary to modify the requirements to achieve the desired functionality [5]. There are many reasons why software must change to accommodate these differences. In fact, it is likely that more than half of the system requirements will change before deployment [6]. Managing customer requirements is one of the key problem areas in software system development and production [6]. Ideally, developers prefer to create a set of requirements that are stable, which is not practical.

Change management is one of the most important aspects of a successful software development project. Manny Lehman aimed to describe common issues concerning software systems that change.In this regard created software evolution laws. Developers must also be aware of the risks associated with changes. Requirements engineering is the basis for software development. A change request is a requirement to add to initial requirements, which also includes a change request related to hardware[2]. Software, regardless of the precision of the development process or the depth of problem understanding by the developers, will change. But, it is often impossible to make all the correct requirements and implementation decisions at the beginning [7]. Hence the risk increases as development progresses.

Because of these divergent changes, the change request analysis should ensure to predict the risks possible with regard to apply the change requirement.

2. RELATED WORK

To assess the impact and risk associated with change requests to software can be classified with currently existing classification schemes. These classifiers mainly classify the impacts due to change requests. The functional aspects of these classifiers can be observed in the literature as

- Determining risks associated with change request and identifying the scope of acceptability of the change request.
- Allowing engineers to group changes based on different criteria such as the cause of the change, the type of change, the location where the change must take place, and the potential impact of the change.
- Allowing engineers to develop a common approach to deal with similar changes, resulting in less overall effort required than if each change was addressed individually [8].

Lientz et al [9] work identified the frequency of the different types of maintenance activities performed by a large sample of software development organizations. Based on their work and work by Sommerville et al [10], the major types of changes related to perfection, correction, adapt and prevent have been identified. Changes related to Perfection are the result from changes adopted during the SDLC process. These changes aimed to advance the system to achieve scalability in

target requirements. Predicted or confirmed defects cause the change requests that demands corrections. Adapting to the new software environment or system platform can be categorized as adaptive [10]. Change requests that aimed to achieve stability in software model against impending problem can categorized as prevention category [11].

Nedstam et al [12] described the process of change request as a process flow:

1. Recognizing a need that is evolved
2. Resource allotment for change request analysis and implementation
3. Assess feasibility and impact of the change request.
4. Define a strategic change request handler
5. Define methods of implementation
6. Initiate change request.

A method that defines artifacts of the change request will be addressed in this paper to assist SDLC crew, in particular in steps 3 and 4. By defining artifacts of the change request it is possible to conceptualize the impact of a proposed change on all other phases of the SDLC.

The model that defines artifacts is build on features of existing change classification and analysis schemes that provide insight into changes that affect software architecture. Kung et al [13] studied the impact of code changes on the class inheritance structure within a software system. Nedstam, et al [18] identified changes that affect the architecture or system process or both.

There are various types of software dependencies. Product metrics regarding direct dependencies are categorized into syntactic dependency, and process metrics regarding rational coupling are categorized into rational dependency [14]. Cataldo et al. [15] compared the strengths of the correlations between various dependencies and faults. Their examination showed that the correlation between syntactic dependency and faults was insignificant or weak and that the correlation between rational dependency and faults was significant and the strongest. The results are one of the cases that product metrics are insufficient for fault prediction in maintenance.

Zimmermann et al [16] applied social network analysis (SNA) on a software dependency graph representing relationships between binary modules of software systems. They reported that adding network measures from SNA literature could improve the performance of fault prediction. Although the network measures are product metrics, they are not covered by the syntactic dependency categorized by Cataldo et al [15].

Kenichi Kobayashi et al [17] assumed that the network measures used by Zimmermann and the rational dependency used by Cataldo [15] share common factors of fault-proneness. Therefore, they assumed that the change impact analysis [18] on source code enables us to extract implicit dependency, such as relations exposed by rational coupling. Change impact analysis is a technique that detects affected areas of source code when some part is changed. In Cataldo's examining the number of rational couplings of a given module was most correlated with faults. Therefore, we expected the scale of estimated areas affected by any changes correlates with fault-proneness as well as the number of rational couplings.

However, it is difficult to compute the exact affected areas of change impact. Static analysis [18] has a nature that it may derive excessive (false positive) areas. Besides, it requires enormous computation time to improve the accuracy. Dynamic analysis [19] can easily capture dynamically bound areas, but it has a nature that it may fail to capture affected areas which are seldom used. In reality, there are many cases in which dynamic analysis cannot be performed. A practical technique to find the affected areas has been proposed for the case where the change to a given module is already known [20]. However, since we need to know the area before the change is given, it is difficult to compute the areas while minimizing false positives.

In the best of our knowledge and annotations done on state of the art in impact analysis of change request, we can conclude that the most of the literature on impact analysis of the change request is aiming to define the impact in developer context, in particular development phase of the SDLC. Hence here we attempted to provide an analytical study to derive the artifacts of change request to assess impact of that request on SDLC phases.

3. DEFINING ARTIFACTS FOR CHANGE REQUEST

As an initiation of our research, we proposed a scheme to define artifacts for change request to assess impact of that change request on structural design of the SDLC phases. This proposed scheme is motivated and using change and defect classifications that identify aspects of an SDLC phases, which would influence by change request considered.

3.1 Overview of the Model that define Artifacts

The model to define artifacts for change request was designed to assess the effects of change request on structural design of the SDLC phases and leads to analyzing the impact of change request on SDLC phases. This proposed model defines the artifacts starting with the high-level features of the change, and then progress to a more detailed selection of desired change requests artifacts and fallows to artifact's effect on SDLC phases. The high-level features describe necessitate of the change request, the change category, the load of the change, the change's impact to the SDLC static and dynamic features, and finally the functional and nonfunctional requirements that will be affected by the change. The detailed artifacts of the change request identify the specific structural design changes that should be made to the major structural design annotations of the SDLC phases in order to implement the change.

The following steps brief the process of defining artifacts for a change requested.

- Recording the mandatory actions essential on SDLC phases in descriptive format.
- Verify the correlation between the actions recorded
- Assess the impact of the actions correlated to verify that change can be implemented in actual constraints or not.
- If implementation of change request is not limited to actual constraints then generate the consensus on impacts of change request on other SDLC phases that helps to recommend the changes required in structural design of the different SDLC phases. These changes are correlated to the actual change request.

The proposed model defined as a decision tree where choices made for the high-level artifacts will affect the decisions taken at various levels of SDLC. The Inter dependability of these artifacts is needed to elaborate, which identifies possible constraints and dependents concern to high-level artifacts that are selected.

3.2 Defining wide-ranging Artifacts:

The top layered artifacts describe the overall characteristics of the change and its effect on the whole system and development environment. Table 1 shows the wide-ranging Artifacts. In the figure describes the top layer artifacts and their possible values. The values for most of these top layered artifacts are measured using the Overall Impact Scale. The developer must first select the need of the change requested. The change request need can be due to an enhancement proposal or defect. The information provided by the motivation artifact is much for the multiple changes, however as time advancements, the occurrence of defects versus enhancements will afford supplementary imminent into system maintenance. A raise in the numeral of defects establishes overtime may clue to a system that is moribund in maintainability as more defects are commenced during the maintenance practice [10]. The next artifact, type, determines the type of change request. The value of this artifact can be ideal, remedial, acclimatize or defensive. The occurrence observed in change request type during maintenance indicates eminence factors affecting core necessities concluded during SDLC such as reflections on system portability due to recurrent acclimatize change requests [22], potential maintainability due to recurrent ideal change requests [22]. The "coarse effect" of the change request as artifact explains the load of the change request in terms of its impact on structural design of the system. The change request load can be identified as functional, architectural or reform. Purely functional changes affect the values of user observable artifacts and functional artifacts. The change requests of the system architecture are affects only the values of architectural artifacts [12]. Reforming usually takes place to satisfy some quality artifacts such as maintainability, suppleness, or complication [23]. One or more types loads can be observed from on change request. The features artifact determines the change request impact on SDLC phases. An inert change request affects the rational system features, such as the decomposition of modules at the design phase, dependency analysis at requirement analysis phase, the inheritance structure defined during modules development phase. A change to the dynamic features affects how the data is propagated through the SDLC phases, in particular, the behavior of distributed components, how prescribed concurrent processes influence, and other expected runtime behaviors. For the "features" artifact we rate the Impact to determine the extent of the effect on each feature. Range values between 0 and 4 that includes these 0 and 4 used to rate the impact of change request on a selected feature. These rates indicate the no impact by 0 and the abnormal impact by 4, which conclude the drastic effect of change request on the features.

Table 1: Taxonomy of Wide range Artifacts

Artifact	Choice of artifact value
Motivation	Enhancement or defect
category	ideal, remedial, acclimatize or defensive
coarse effect	functional, architectural or reform
Features	static, dynamic
Logical	Dependency , Layers, Module Decomposition, Source Structure, Inheritance Structure
Runtime	control flows, repository access, concurrency, components, distribution, deployment
Non – Functional	usability, reliability, availability, security, portability, complexity, flexibility, scalability
Functional	Technology, interfaces, data access and transfer, environmental,

Further discussed top layered artifacts identify that which software engineering issues the change request addresses in terms of functional and non-functional requirements. As of the process of the model, Assessing feasibility and impact of the change request, rank the impact of the change request in between 0 to 4 on features defined under functional and non functional artifacts. This rating helps to identify the constraints to apply change request. The values of the top layered artifact related functional issues were derived from several sources in the literature that examined software change. These values of the functional artifacts include the functions of data such as access and transfer, system interface, system environment, user level interface, domain specification limits, and discretionary others attribute that allows for adding additional issues not currently addressed by the scheme [21], [24], [25], [26], [27]. The values of the top layered artifact "non-functional issues" elevates about ability of usage, reliable or not, availability constraints, security constraints, also includes the portability issues, intricacy, suppleness, scalability, and elective other features that allow for additional non-functional requirements not listed currently. The final set of wide ranging artifacts offers more detail into the changes that must be made to the structural design defined by SDLC phases. The developer can choose the rational and runtime architectural annotations that must be changed in order to implement the change request. The top layered artifact "rational" includes a comprehensive list of general structural design properties that can be used to describe the framework derived from SDLC phases in particular, most object oriented software intensive systems. The other top layered artifact "runtime" serves a similar purpose as the artifact "rational". This artifact list the dynamic structural design properties defined during SDLC phases, in particular, common to most object oriented software intensive structural design. The Overall Impact Scale will be used to assess the impact of the change request on processing of control flow , accessing

resources repositories, processes that are concurrent, interactions between components, component distribution and deployment of component.

The wide range artifacts aim to describe the overall impact of change request on SDLC phases. The explicit-range artifact which provides more details into the change requests to the rational and runtime structures is fallows.

3.3 Explicit-Range Artifacts

The explicit range artifacts allows to analyze the structural design while making recommendations for changes to the overall structure in order to implement the change request.

The changes that are reflected in the architecture include changes to any structural design module, interface, component, and connector. A specific impact rating strategy used to describe the magnitude of the changes that can be made to the structural design defined by SDLC phases. Each rating will correspond to the type of correlated change applied to an item in the rational and runtime lists of structural design.

The rational and runtime artifacts of the system contain several static and dynamic annotations. An annotation is a depiction of a set of system elements and the relationships among those elements. Both the rational and runtime artifacts presented in the structural design change scheme describe several different aspects of SDLC phases, in particular Object oriented system. The goal in producing the explicit-range artifacts was to create a comprehensive list of wide range artifacts that can be used to describe the types of changes that could be made to any SDLC phase.

The rational and runtime artifacts together referred as explicit range artifacts of the system contain several static and dynamic system annotations. An annotation is a representation of a set of system elements and the relationships among those elements. Both the rational and runtime artifacts presented in the structural design change scheme describe several different aspects of SDLC phases, in particular Object oriented system. The goal in creating the explicit range artifacts was to create a comprehensive list of wide range artifacts that can be used to describe the types of changes that could be made to any SDLC phase.

The list of rational artifacts includes a description of the types of changes that can be made to elements of any annotation that exhibits explicit range artifacts. Table 2 provides an overview of the specific explicit range artifacts and the types of changes that can be made to elements in each structural design annotation. These changes could include adding, update, and delete and connect can apply to elements. The rational annotations range of modules in layered structure to individual module and all relationships in between.

The "dependency" artifact describes changes that are made to modules that affect their relationships with other modules that they depend on. These changes include adding deleting or update dependencies between modules. The impact rating allotted for the applied changes would determine the severity of the actions performed on any dependency modifications.

The "stratum" artifact identifies the modifications to the elements in a structural design annotation that shows how the system divided into its various layers. Then a decision can be taken that one or more layers should be modified to carry out the change. The size and complication of any adapted layers would be noted by the value selected in the Specific Impact rating between 0 and 4. If multiple layers are changed then the estimates are going to require a great deal of effort to implement, a value of ‘4’ would be suggested for the Layer change type. The other changes that could be made with an annotation having layers include adding an internal module to a layer; add a connection between two modules within a layer and/or between two modules of different layers.

The "inheritance" artifact addresses changes made to annotations that depict inheritance relationships between modules. Changes that applied to annotations showing inheritance relationships, which could include adding a child module to an existing parent and adding a parent module with new children. The developer may possibly decide to modify or remove an existing parent-child hierarchy. Hence there could be a change occurrence at functional or interface levels of the parent child modules.

The "decomposition" artifact includes annotations about relationships between system modules. The changes that applied to these annotations includes adding a module, modify a module, remove a module and alter modules relation.

The final rational structural design change artifact is the "code". The "code" artifact allows to record changes to strategy of storing source code on a computer that contains system modules. This artifact registers the changes occurred to specific files and packages on the system. It also records the changes occurred at an external library.

The runtime artifacts that part of explicit range artifacts explain potential changes that could be made to runtime structural design annotations. These annotations will include system dispensation apparatus and the connectors amid these components. There are various illustrations of the processing components of any SDLC phase, in particular, object oriented system. The proposed process of defining artifacts provides an inclusive list of these representations and the possible changes that could be made to them.

The "control flow" artifact describes changes that could be made to annotate that exhibit qualities of a pipe-and-filter or batch sequential style. These correlated changes include adding and update the functionality of the processing units, update the format or values that the processing units process, update a connection that connects any of two process units, and adaptation to interfaces that used by a processing unit to access external data.

The "repository" artifact identifies changes made to any annotation that includes a shared data storage. The kinds of users that have access to this data storage can be altered and then facilitate to conclude the changes to a user’s authorization credentials, the change occurrence at accessibility of data from a user and adding a data storage to a system.

The "concurrent" artifact illustrates component processes that communicate and execute in a concurrent manner. The annotations about changes occurred to the components that involved in concurrent process indicates the changes desired to that processing unit, the information shared between any of the processing units, or a change in the connection strategy of any two units.

The "interaction" artifact contains viewed involved in implicit invocation or publish-subscribe structural design representations. This artifact handles changes to the event driven structural designs. The components in a component interaction annotation could broadcast events to other components or a component that listens the events

broadcasted. The modification occurred to components that participating in interactions will be recorded by this artifact, also records the updates applied to an event broadcasted, updates applied to event registration process between any of the components.

The "distributed" artifact highlight changes to distributed structural designs. These distributed structural designs involve remote components that interact by transferring data and/or allowing access to specified non local objects. A peer-to-peer or client-server connectivity are the considerable format to represent distributed relations. As an example, the developer must consider the connection type between peer to peer need to be checked while considering structural design related change requests. Where in the case of client to server, the location of the resources such as the client interface, application server and data storage would be considerable to modify and record of the developer.

The "deployment" artifact maps processing units to hardware in regard of a hardware change request or to the location in regard of a processing change request.

Table 2: Taxonomy of Explicit range artifacts

Specification	Artifact	Values
Runtime	Control flow	Add, delete or update events on · processing unit · input, output of processing unit · format of input output · connectionsexternal interface
	Repository	Add operation on user type Add, update or delete operation on Access privileges Attachment Repository
	Concurrent	Add, update and delete operations on Concurrent processes Synchronize processes Connect process Data exchange
	Interaction	Add, update or delete operation on Publicize a component Event Publicize an event record an event Listen to an event
	Distributed	Add, delete or update operation on Peer Client Type Server Component Connection Update operation on layer structure
	Deployment	Add, update or delete operation on Hardware Process Location
Rational	Dependency	Add, Update or Delete operation on dependency
	Stratum	Add, Update or Delete Operation on Stratum Intra and inter stratum connection Stratum module
	Inheritance	Add, Update or delete operation on Parent Child Interface Interface Connection
	Decomposition	Add, Update or Delete operation on Module Relation
	Source	Add, Update or Delete Operations on Package structure File Structure Library usage Structure

4. CONCLUSION

The model proposed here is defining artifacts to provide as an input to any of the classifier such as decision tree, SVM that incorporates change request classification, impact analysis, and predicts correlated changes in different SDLC phases, Also assess risks to aid SDLC crew in making decisions for changes based on how the system will be affected. We continue to refine the proposed model that defines artifacts for change requests. We use these models to define artifacts for change requests from historical data sets and correlate those changes to the implementation data to assess correlation changes. Finally, we projected to attainability further analysis on the data to examine an information-theory based metric approach to measure consensus when multiple values assigned to predicted artifacts under various classification models.

5. REFERENCES

[1]. M.M. Lehman and L. Belady, Software Evolution - Processes of Software Change, Academic Press, London, 1985

[2]. L. Arthur, Software Evolution: The Software Maintenance Challenge, John Wiley & Sons, Toronto, Canada, 1988.

[3]. C. Jones, "Software Change Management," Computer, vol. 29, no. 2, 1996, pp. 80-82.

[4]. B. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[5]. G. Stark, A. Skillicorn, and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Releases " Crosstalk: The Journal of Defense Software Engineering, vol. 11, no. 12, 1998, pp. 11-16.

[6]. G. Kotonya and I. Sommerville, Requirements Engineering: Processes and Techniques, John Wiley & Sons, Chichester, West Sussex, England, 1998.

[7]. J. Siddiqi, "Requirement Engineering: The Emerging Wisdom," IEEE Software, vol. 13, no. 2, 1996, pp. 15.

[8]. N. Nurmuliani, D. Zowghi, and S. P. Williams. "Using Card Sorting Technique to Classify Requirements Change," in Proceedings of the 12th IEEE International Requirements Engineering Conference, 2004, pp. 240-248.

[9]. B. Lientz and B. Swanson, Software Maintenance Management Addison-Wesley, 1980

[10]. I. Sommerville, Software Engineering. 7th ed: Addison-Wesley, 2004

[11]. P. Mohagheghi and R. Conradi. "An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality Vs. Quality Attributes," in Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04), 2004, pp. 7- 16.

[12]. J. Nedstam, E. A. Karlsson, and M. Host. "The Architectural Change Process," in Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04), 2004, pp. 27-36.

[13]. D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen. "Change Impact Identification in Object Oriented Software Maintenance," in Proceedings of the International Conference on Software Maintenance, Victoria, BC, 1994, pp. 202-211.

[14]. Gall, H., Hajek, K., and Jazayeri, M., "Detection of rational coupling based on product release history," IEEE Int'l Conf. on Softw. Maint. ICSM, pp.190-198, 1998.

[15]. Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D., "Software dependencies, work dependencies, and their impact on failures," IEEE Trans. Softw. Eng. 36, 2, pp.864-878, 2009.

[16]. Zimmermann, T., and Nagappan, N., "Predicting defects using network analysis on dependency graphs," Int'l Conf. on Softw. Eng. ICSE, pp.531-540, 2008.

[17]. Kobayashi, K.; Matsuo, A.; Inoue, K.; Hayase, Y.; Kamimura, M.; Yoshino, T.; , "ImpactScale: Quantifying change impact to predict faults in large software systems," Software Maintenance (ICSM), 2011 27th IEEE International Conference on , vol., no., pp.43-52, 25-30 Sept. 2011; doi: 10.1109/ICSM.2011.6080771

[18]. Bohner, S. A., and Arnold, R. S. (Eds.), "Software change impact analysis," Bohner, S. A. and Arnold, R. S., "An introduction to software change impact analysis," IEEE Computer Society Press, pp.1-26, 1996.

[19]. Grove,D., and Chambers,C., "A framework for call graph construction algorithms," ACM Trans. Program. Lang. Syst. 23, 6, pp.685-746, 2001.

[20]. Law, J., and Rothermel, G., "Whole program path-based dynamic impact analysis," Int'l Conf. on Softw. Eng. ICSE, pp.308-318, 2003.

[21]. Ren, X., Shah, F., Tip, F., Ryder, B. G., and Chesley, O., "Chianti: a tool for change impact analysis of Java programs," Conf. on Object-Oriented Prog., Syst., Lang., and App. OOPSLA, pp.432-448, 2004.

[22]. J. Bosch, Design and Use of Software Architectures: Addison Wesley, 2000

[23]. V. Basili and D. Weiss. "Evaluation of a Software Requirements Document by Analysis of Change Data," in Proceedings of the 5th international conference on Software engineering, San Diego, CA, IEEE Press, 1981, pp. 314-323.

[24]. L. C. Briand and V. R. Basili. "A Classification Procedure for the Effective Management of Changes During the Maintenance Process," in Proceeding of the Conference on Software Maintenance, Orlando, FL, 1992, pp. 328-336.

[25]. N. H. Madhavji. "The Prism Model of Changes," in Proceedings of the 13th International Conference on Software Engineering Austin, TX, 1991, pp. 166-177.

[26]. A. Mockus and L. G. Votta. "Identifying Reasons for Software Changes Using Historic Databases," in Proceedings of the International Conference on Software Maintenance, San Jose, CA, 2000, pp. 120-130.