IJDCST @Feb-March Issue- V-2, I-3, SW-06
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Speculative Analysis Exploits Qualitative and Quantitative User Studies

[1] Hemalatha Narne ,[2] Adepu Sridhar.
[1]Vignan university,Vadlamudi, Gutur.
[2]Assistant Professor , Vignan university,Vadlamudi, Gutur.

ABSTRACT: Identifying and resolving development processes of sharing project by software developers arise in collaborative development and can slow progress and decrease quality of the assurance development of shared project. Traditionally developed and design crystal tool as a speculative analysis in real time application development. Crystal, a publicly available tool that helps developers identifies, manage, and prevent conflicts. Crystal uses speculative analysis to make concrete advice unobtrusively available to developers. Qualitative and Quantitative approaches have typically been combined by using them side-by-side or sequentially, until the point when the separately generated results are interpreted and conclusions drawn. In this we propose to develop a mixed method to describe the analysis of the qualitative and quantitative methods. The mixed methods research purpose most frequently served by integration of analyses is initiation, that is, to be provocative and bring fresh perspectives through contradiction and (intended or unintended) discovery of paradox. Experimental results show efficiency of the conflicts and risks present in the shared project development.

KEY WORDS: Crystal Tool, Qualitative and Quantitative approaches

I. INTRODUCTION

Software Engineering is the study of design, development and maintenance of software. In other words it is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software and an engineering discipline that is concerned with all aspects of software production. Communication skills, team dynamics, working with a "customer," and creativity are also important factors in the software engineering. It is important because of the large expensive software systems.

Software development process:

A set of activities that leads to the production of a software product is known as software process. Computer-aided software engineering (CASE) tools are being used to support the software process activities. As there is a vast diversity of software process for effectiveness and limited case tools and different types of products. There is no ideal approach that has yet been developed to software process. There are some fundamental activities that are common in all process activities and some of them are like software maintenance, design, validation and specification.

A software development process is also known as software development life cycle (SDLC). It is a term used to describe a process of analysis,

planning, design, maintenance, deployment and implementation of an application.

Risk Management in Software Engineering:

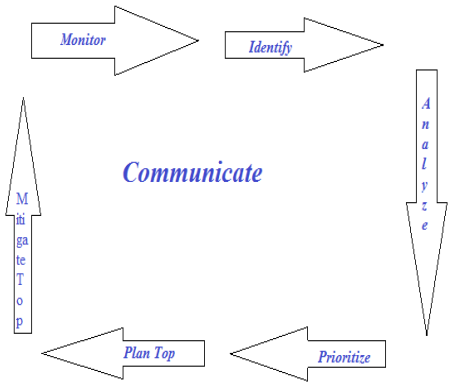
The management of a risk is the important future in throughout the software development life cycle. A risk is a potential future harm that may arise from some present action, such as, a schedule slip or a cost overrun.

"Risk in itself is not bad; risk is essential to progress, and failure is often a key part of learning. But we must learn to balance the possible negative consequences of risk against the potential benefits of its associated opportunity."

Risk management is a series of steps whose objectives are to identify, address, and eliminate software risk items before they become either threats to successful software operation or a major source of expensive rework.

The Risk Management Process:

The risk management process can be divided into two phases. Those are risk assessment and risk control. The risk assessment further broken down into risk identification, risk analysis, and risk prioritization. Like that risk control also divided into risk planning, risk mitigation, and risk monitoring.



Fig(1). The Risk Management Cycle

Software Risk Management:

There could be risk associated with the every software project, the mail goal is to identify and manage those risks. The most important risk management tasks are risk index, risk analysis, and risk assessment.

1).Risk Index: Risk index is the multiplication of impact and probability of occurrence. Risk index can be characterized as high, medium, or low depending upon the product of impact and occurrence. Risk index is very important and necessary for prioritization of risk.

2).Risk Analysis: The risk analysis is used to identify the high risk elements of a project. The main purpose of risk analysis is to understand risks in better ways and to verify and correct attributes. A successful risk analysis includes important elements like problem definition, problem formulation, data collection.

3).Risk Assessment: It integrates risk management and risk analysis. . Risk assessment requires correct explanations of the target system and

all security features. It is important that risk deferent levels like performance, cost, support and schedule must be defined properly for risk assessment to be useful.

Strategies for Risk Management:

Throughout the software development process, there are various strategies for risk management could be identified and developed. The risk strategies could be divided into three classes namely careful, typical, and flexible. Generally, careful risk management strategy is proposed for new and inexperienced organizations whose software development projects are connected with new and unproven technology; typical risk management strategy is well-defined as a support for mature organizations with experience in software development projects and used technologies, but whose projects carry a decent number of risks; and flexible risk management strategy is involved in experienced software development organizations whose software development projects are officially defined and based on proven technologies.

II. RELATED WORK

- Measuring the effect of conflict on software engineering teams by J. S. KARN AND A. J. COWLING.
- Effects of intra-group conflict on packaged software development team performance by Steve Sawyer.
- Resolving conflicts in requirements engineering by Camilo Fitzgerald.
- Software Errors And Complexity: An Empirical Investigation Victor R. Basili And Barry T. Perricone.
- Common Errors in Large Software Development Projects by David A. Gaitros.

The identification of the various factors that have an effect on software development is of prime concern to software engineers. The specific focus of this paper is to analyze the relationships between the frequency and distribution of errors during software development, the maintenance of the developed software, and a variety of environmental factors. These factors include the complexity of the software, the developer's experience with the application, and the reuse of existing design and code. Such relationships can provide an insight into the characteristics of computer software and the effects that an environment can have on the software product. Such relationships can also improve the reliability and quality with respect to computer software. In an effort to acquire knowledge of these basic relationships, change data for a medium-scale software project were analyzed.

Developing software is a relatively new area of enterprise that bears little resemblance to other engineering disciplines. Although the term software engineering is widely used throughout the business, the act of creating a new piece of software can hardly be compared to the design and construction of a new building or bridge. Computer scientists are still struggling after 30 years to define software engineering and to find the right combination of techniques, procedures, and tools that assure success in development of large complex systems.

Conflict:

Conflict is a natural disagreement resulting from individuals or teams that differ in attitudes,

beliefs, values, or needs. As human beings interact in organizations, differing values and situations create tension. One prominent scholar of conflict listed the following issues involved in conflicts:

- Control over resources;
- Preferences and nuisances in which the tastes or activities of one party impinge upon another;
- Values, when there is a claim that a value or set of values should dominate;
- Beliefs, when there is a dispute over facts, information, reality, and so forth;
- The nature of the relationship between the parties.

The traditional view of conflict was that it was a negative phenomenon and a serious threat to effective team performance. However, this is not a universally held opinion amongst conflict researchers, and it has been challenged: Scholars have argued that more focus should be placed on the form the conflict takes. The point is that conflict per se need not be a negative force. Indeed, some have argued persuasively that when positive conflict is recognized, acknowledged, and managed in a proper manner, personal and organizational benefits can accrue.

Constructive and Destructive Conflict:

Conflict has been given a bad name by its association with disruption. However, as was mentioned in the previous section, several researchers have argued that conflict need not be a negative force and that it is often the case that it is the form the conflict takes that determines how much damage is caused that is, whether it is a constructive or destructive conflict.

Constructive conflict is characterized by cooperation and flexibility. The principal focus is on trying to achieve a solution between struggling parties that is mutually satisfactory to everyone. However, destructive conflicts are more concerned with power struggles and personal antagonisms and are characterized by domination, escalation, retaliation, com petitions, and inflexibility. When a conflict spirals out of control, it runs the risk of becoming destructive. When this happens, participants lose sight of their initial goals and focus on hurting the adversary.

Persistence of Conflicts:

To maintain a relationship longer time it has to change more opportunities into more severe relationship. By using history we have to trace the backward in time to measure the lifespan of a conflict from the two change sets. When two branches came in to conflict with each other, those sets were merged to find the earliest point in time. For this purpose we have to create a time order list of the change sets from each of the two branches. It coexisted at each point in time to see if they were in conflict, stopping when we found a no conflicting pair. This approach compressed all other sub branches and merges that existed on the branches that contributed to the merge under analysis.

Before it has been resolved the textual X relationship is persisted up to 3.2days and involved 18 .3 change sets in average case. The developers let

know about the details of TEXTUAL relationships immediately upon their creation. In the worst case, one TEXTUAL X relationship in MaNGOS persisted for 334 days and included 676 changesets by one of its developers before it was resolved.

In order to prevent future conflicts, a developer has to know about that they can measure other changes safely. To maintain a textual relationship persistently, the more opportunities it has to change into a conflict. Accordingly, we asked "How long do developers experience the TEXTUALp relationship?" We measured the lifespan of a TEXTUALp relationship for each conflict-free merge in the history.

Before incorporation, the TEXTUALp relationship persisted for 2.4 days and involved 12.7 change sets (with median values of 0.8 days and 7 change sets) in average case. The developers have to learn immediately about the textual relationship and encouraging earlier and smooth incorporation, A tool can be helped. In the worst case, in terms of time, one TEXTUALp relationship in Voldemort persisted for 138 days; in terms of change sets, one TEXTUALp relationship in Gallery3 persisted for 232 change sets without a merge, while each of the possible merges along the way would have been textually clean and fully automated. Neither of these two long-lived TEXTUALp relationships evolved into a conflict.

system	TEXTUAL✗ relationships							TEXTUAL✓ relationships						
	#	length (days)			length (changesets)			#	length (days)			length (changesets)		
		mean	stddev	median	mean	stddev	median		mean	stddev	median	mean	stddev	median
Voldemort	28	7.5	9.9	2.1	23.2	33.1	6	139	6.0	13.6	2.7	16.0	35.4	7
Gallery3	47	1.8	6.9	0.1	20.8	70.0	4	459	1.1	6.9	0.6	11.2	36.3	6
Insoshi	23	7.3	14.7	1.9	24.3	44.4	6	70	4.9	13.4	1.1	10.8	13.1	6.5
jQuery	1	5.4	0	5.4	13.0	0	13	17	1.1	2.1	0.4	4.9	3.7	4
MaNGOS	81	1.4	1.5	0.8	13.5	14.0	9	113	2.4	2.8	1.5	16.7	19.6	11
Total	180	3.2	8.0	0.7	18.3	42.6	6	797	2.4	7.7	0.8	12.7	32.5	7

III. EXISTING SYSTEM

Crystal precisely reports actual conflicts, determining the relationship between two developers' states by actually creating the merged artifact. In other words, to find out what would happen if Bill and Melinda merged their code, Crystal, in the background, makes a copy of Bill's code and incorporates Melinda's changes. Similarly, once Crystal creates the merged code artifact, it attempts to compile and to execute the test suite on that artifact. Again, Crystal only reports a compilation or testing conflict when the build or a test actually fails. Because the computation happens in the background, the developers can continue to work without interruption; in certain situations, we expect the developers to ignore Crystal, much as they sometimes ignore project bulletin boards and email.

We refer to the idea of attempting a set of actions on the developer's state in the background and reporting on the outcomes of those actions as speculative analysis.

Awareness tools notify developers when they might have conflicting changes. This approximation is computed differently in various tools. Some determine if a co-developer is working in the same file, some report any change to the repository others report concurrent changes to the AST. These approaches can lead to the inclusion of false positives — reporting potential conflicts that do not evolve into actual conflicts. Furthermore, few current awareness tools try to automatically detect higher-order merge conflicts; again, Crystal is precise as it uses the project's tool chain to dynamically detect conflicts by execution of the build system and

test suites. We refer the reader to for a more detailed description of related work.

Crystal can, in rare situations, also report false positives. Check pointed changes that are later discarded can cause a teammate to see a pending conflict that later disappears. This can happen when a developer checkpoints exploratory code or a partial change.

Crystal unobtrusively reports four kinds of information: the developer's local state, relationships with other developers or repositories, the possible actions (which is derived from the local state and relationship with the master repository, and which we omit from this paper for brevity), and guidance about those actions. The remainder of this section summarizes Crystal's interface and the information it reports; more details on both are available.

Example Crystal use

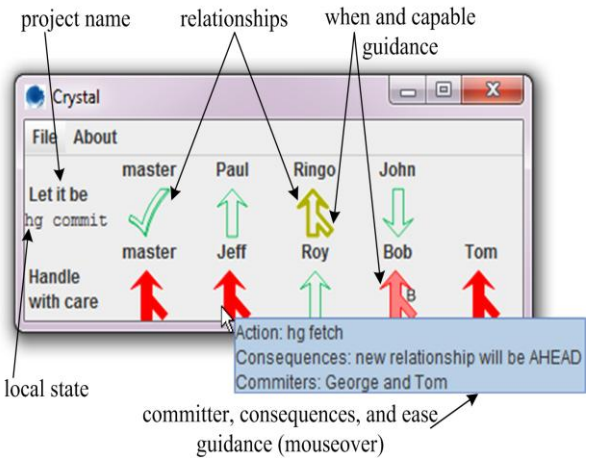


Figure shows a screenshot of Crystal's main window. The window displays a row of icons for each of a developer's projects. In this example, there are two projects: "Let it be" and "Handle with care". The former has four collaborators: George (the developer running Crystal), Paul, Ringo, and John. The latter has five collaborators: George, Jeff, Roy, Bob, and Tom. Each developer can independently choose whether or not to run Crystal.

On the left-most side of each row, underneath the project name, Crystal displays the local state. This tells George, in the native language of the underlying VCS, whether he must checkpoint changes (hg commit, in Mercurial) or resolve a conflict. Then, for each repository (master and other collaborators', whether or not they are running Crystal), Crystal displays the relationship with that repository. If George has the ability to affect a relationship now, the icon is solid, which combines the When and Capable guidance. If George cannot affect the relationship, the icon is hollow. If the relationship is of the might variety—George might or might not have to perform an operation to affect the relationship—the icon is solid but slightly unsaturated (see the relationship with Bob in the "Handle with care" project).These features allow George to quickly scan the Crystal window and identify the most urgent issues — the solid red icons — followed by other, less severe icons. George can also quickly identify whether there is something he can do now to improve his relationships (in the example, George can perform actions to improve his relationships in the "Handle with care" project, but not in "Let it be"), and whether there are unexpected conflicts George may wish to communicate with others about. Holding the mouse pointer over an icon displays the action George can perform and the Committer, Consequences, and Ease guidance, when appropriate.

IV. PROPOSED SYSTEM

By combining the qualitative and quantitative approaches, the points until the separately generated results are interpreted. It will show efficiency of the conflicts and risks that are presented.

Qualitative and Quantitative Approach:

The multiple research methods and tools of qualitative experimental and non-experimental are essential for researchers. The quality of a program is limited when we are not using by combining both the methods. The elements of qualitative and quantitative approaches are combined into a unique design to undertaking as a mixed method. The way in which the mixed methods might be differentiated at which the elements of qualitative and quantitative approaches are integrated together. The purpose of using both methods to finding the corroborative evidence from different methods.

Strategies for Integration:

There are four strategies for combing of both qualitative and quantitative approach. Those are: (a) Data Transmission (b) Typology Development (c) Extreme Case Analysis (d) Data Consolidation.

Data transmission: The one form of data is transformed into other form is known as data transmission.

Typology Development: Classification of data from one set of data is applied to another set is known as typology development.

Extreme Case Analysis: The outliers or residuals revealed by one analysis are explored using alternative data or methods are known as Extreme Case Analysis.

Data Consolidation: To create variables for use in further analysis is known as data consolidation.

Combination of mixed method analysis is most obvious when data from one type is used in analysis of other type. The strategies of integration might be used in the context of expansion, development and complementarity. But the integration with corroboration is inconsistent. The popular association of mixed methods with corroboration and consequent lack of consideration of integrative strategies; and the view that integration or synthesis of results is an intellectual or ideologically driven activity. To achieve integration of data analysis, it requires the capacity to visualize what might be possible to set the new paths. Integration is greatly helped by data handling technology to facilitate the process

Two Major Routes to Integration in Analysis:

Propose in terms of data handling, there are two major routes to integration that underlie the various strategies are

1). Combination of data types with in an analysis, which is used for both statistical analysis and comparison of coded narrative material. This could occur through by combining both numerical and textual data. For example combination of survey and interview.

2). Conversation of data from one type to another type for analysis. The conversation of qualitative codes to codes used in a statistical

analysis through the contribution of qualitative analysis.

Using Software to Combine Numeric and Text Data for Analysis:

The data management is to combine mixed forms of data and procedures for working with them. The advent of text-handling spreadsheets and databases and, in particular, of text analysis software, has heralded solutions to these data management problems, and opened up new possibilities for more rigorous and/or deeper analysis of this type of data. They have not necessarily solved the theoretical issues which could arise when different forms of data are combined.

V. EXPERIMENTAL RESULTS

Speculative analysis over version control operations provides precise information about pending conflicts between collaborating team members. These pending conflicts—including textual, build, and test—are guaranteed to occur (unless a developer modifies or abandons a committed change). Learning about them earlier allows developers to make better informed decisions about how to proceed, whether it is to perform a safe merge, to publish a safe change, to quickly address a new conflict, to interact with another developer and so on. Our retrospective, quantitative study of over 550,000 development versions of nine open-source systems, spanning 3.4 million distinct (and a total of over 500 billion, over all versions) NCSL, indicates that

1. conflicts are the norm rather than the exception,

2. 16 percent of all merges required human effort to resolve textual conflicts,

3. 33 percent of merges that were reported to contain no textual conflicts by the VCS in fact contained higher-order conflicts, and

4. Conflicts persist, on average, for 3.2 days (with a median conflict persisting 0.7 days).

A range of statistical techniques, including several based on patterns of association, are being used in an ongoing concept analysis of research performance (Bazeley, unpublished data). The primary data comprise descriptions given by 295 academics for eight different aspects (‘brands’) of research performance—descriptions of researchers who are productive, active, recognized, satisfied, approachable, and/or who demonstrate quality, ability, benefit. These have been coded using NVivo to create a set of descriptors. Additionally, basic demographic data are available, along with each academic respondent’s weighting of the importance (or value) of each of these eight aspects of performance for doing research and for assessing research (as interval scales). These additional numeric data have been imported into the NVivo database for use in combination with text responses, and coding based on the descriptions given has been exported from NVivo in a number of forms, each contributing to a different type of analysis.

These techniques are all being used in an exploratory way, appropriate to the purpose of exploring and elucidating a concept. Extensions to this work are likely to involve confirmatory strategies.

VI. CONCLUSION

To date, it is developments in software programs for analysis of qualitative data that have contributed most noticeably to researchers' capacity for integrating methods in the ways described in this paper. Indeed, Lyn Richards has argued that the most radical methodological changes that came about with qualitative computing were not in what the computer could do (such as coding), so much as the uses to which it could be put in driving a complex and iterative data interrogation process.

Tools are still being developed, a process which is both responsive to and which can lead to new techniques in data analysis. So, these techniques are used to overcome the conflicts in the existing system and as a result we get the effective results by using the techniques used in this paper.

Future Scope:

As further improvement of our application achieves a systematic representation of both qualitative and quantitative analysis using some newly developed tools specifications like RASOOL, ATUSA, etc. This improvement gives better results compared to earliest tool generation in finding conflicts and risks.

VII. REFERENCES

[1] B. Al-Ani, E. Trainer, R. Ripley, A. Sarma, A. van der Hoek, and D.Redmiles, "Continuous Coordination within the Context of Cooperative and Human Aspects of Software Engineering," Proc. Int'l Workshop Cooperative and Human Aspects of Software Eng., pp. 1-4, May 2008.

[2] B. Appleton, S.P. Berczuk, R. Cabrera, and R. Orenstein, "Streamed Lines: Branching Patterns for Parallel Software Development,"Proc. Pattern Languages of Programs Conf., 1998.

[3] T. Ball, J.-M. Kim, A.A. Porter, and H.P. Siy, "If Your Version Control System Could Talk," Proc. Workshop Process Modelling and Empirical Studies of Software Eng., May 1997.

[4] J.T. Biehl, M. Czerwinski, G. Smith, and G.G. Robertson, "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams," Proc. SIGCHI Conf. Human Factors in Computing Systems, pp. 1313-1322, Apr. 2007.

[5] C. Bird, P.C. Rigby, E.T. Barr, D.J. Hamilton, D.M. Germa´n, and P.T. Devanbu, "The Promises and Perils of Mining Git," Proc. Sixth IEEE Int'l Working Conf. Mining Software Repositories, pp. 1-10,2009.

[6] C. Bird and T. Zimmermann, "Assessing the Value of Branches with What-If Analysis," Proc. ACM SIGSOFT 20th Int'l Symp. Foundations of Software Eng., 2012.

[7] Y. Brun, R. Holmes, M.D. Ernst, and D. Notkin, "SpeculativeAnalysis: Exploring Future States of Software," Proc. FSE/SDP Workshop Future of Software Eng. Research, pp. 59-63, Nov. 2010.

[8] Y. Brun, R. Holmes, M.D. Ernst, and D. Notkin, "Crystal: Precise and Unobtrusive Conflict Warnings," Proc. 19th ACM SIGSOFT Symp. and 13th European Conf. Foundations of Software Eng., Sept.2011.

[9] Y. Brun, R. Holmes, M.D. Ernst, and D. Notkin, "Proactive Detection of Collaboration Conflicts," Proc. 19th ACM SIGSOFT Symp. and 13th European

Conf. Foundations of Software Eng., pp. 168-178, Sept. 2011.

[10] M. Cataldo, P.A. Wagstrom, J.D. Herbsleb, and K.M. Carley, "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," Proc. 20th Anniversary Conf. Computer Supported Cooperative Work, pp. 353-362, Nov. 2006.

[11] "How Distributed VersionControl Systems Impact Open Source Software Projects," C. Rodriguez-Bustos and J. Aponte.

[12] Risk Management in Software Engineering By Sunil Sapkota in adavanced Software Engineering 10-20-2011.

[13]. Crystal: Precise and Unobtrusive Conflict Warnings by Yuriy Brun , Reid Holmes , Michael D. Ernst , David Notkin.