International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-2, December 2012

An Integration of JSD, GSS and CASE Tools towards the Improvement of Software Quality

Dillip Kumar Mahapatra, Tanmaya Kumar Das, Gopakrishna Pradhan

Abstract—The increasing demand of software products for different business organization and individuals day-by-day enforces the developers to use policy, technology in a planned manner for the development of quality software products. It is important to entertain all different phases of software development life cycle (SDLC) i.e. from requirements to implementation, maintenance to re-engineering with the use of integrated computer-aided software engineering (CASE) tools and the use of group support systems (GSS) and joint application development (JAD) in the context of CASE environments to facilitate the entire development process. An integrated framework is proposed that facilitates the developers to build up confidence for the improvement of quality for software products.

Keywords: Software process, Joint Application Development, Group Support System, CASE, Software Quality

I. INTRODUCTION

Almost everyone has an idea about the meaning of *quality*. However, when it comes to quality in the real world, i.e. in conjunction with a software development project, disagreements between the persons involved often lead to further problems. Especially in the case of customer complaints about faults in a software product, it seems to be unclear not only what the requirements are, but also if the software has the "right" characteristics with regard to these requirements.

Quality has always been a difficult topic to define, and software quality has been exceptionally difficult. The reason is that perceptions of quality vary from person to person and from object to object. For software quality for a specific application, the perceptions of quality -differ among clients, developers, users, managers, software quality personnel, testers, senior executives, and other stakeholders.

Many people are involved in developing a complicated software project because no one possesses all the knowledge required. Domain-specific knowledge about an application and its environment as well as technical knowledge about systems development has to be incorporated into a cohesive framework to create a good software requirements specification (SRS). The elicitation of requirements is a collaborative effort among managers, users, and system analysts. One of the major factors that causes misunderstandings in the systems development process is the complexity of the communication and decision making among members of a project team, choosing appropriate design strategy, coding, testing, implementation and maintenance of software products to retain with quality.

Since the development of information systems has long been recognized as a joint effort among systems developers, users, and managers, the need to support collaborative activities during the application development process is evident. Though CASE tools have been used in various stages during the systems development life cycle to improve software productivity, there is no direct support in most CASE tools for direct interactions among project team members. However, group support systems (GSS), an emerging technology designed to facilitate human interactions, may be used separately or in combination with joint application development (JAD), a facilitated systems development methodology, to support collaboration among project team members in the applications development process. The purpose of this article is to explore ways to integrate GSS and JAD with CASE tools to improve the software quality.

II. SOFTWARE QUALITY

Almost everyone has an idea about the meaning of *quality*. However, when it comes to quality in the real world, i.e. in conjunction with a software development project, disagreements between the persons involved often lead to further problems. Especially in the case of customer complaints about faults in a software product, it seems to be unclear not only what the requirements are, but also if the software has the "right" characteristics with regard to these requirements.

Quality has always been a difficult topic to define, and software quality has been exceptionally difficult. The reason is that perceptions of quality vary from person to person and from object to object. For software quality for a specific application, the perceptions of quality -differ among clients, developers, users, managers, software quality personnel, testers, senior executives, and other stakeholders.

The perceptions of quality also differ among quality consultants, academics, and litigation attorneys. Quality in general and software quality in particular have been elusive and hard to pin down is because the word "quality" has many nuances and overtones. For example, some of the highlighted attributes of quality can be as outlined below.

- *Elegance or beauty* in the eye of the beholder
- *Fitness of use* for various purposes
- *Satisfaction of user requirements*, both explicit and implicit

Manuscript published on 30 December 2012.
***** Correspondence Author (s)
Dillip Kumar Mahapatra*, Associate Professor, Head of Department of Information Technology, Krupajal Engineering College, Bhubaneswar, Odisha, India / Biju Pattanaik University of Technology, Rourkela, Odiaha, India.
Tanmaya Kumar Das, Department of Computer Science & Engineering, C.V. Raman College of Engineering, Bhubaneswar, Odisha, India / Biju Pattanaik University of Technology, Rourkela, Odiaha, India.
Gopa Krishna Pradhan, Former Professor, Deptt, of Computer Science & Engg. SOA University, Bhubaneswar, Odisha, India.

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an open access article under the CC-BY-NC-ND license http://creativecommons.org/licenses/by-nc-nd/4.0/

- *Freedom* from defects, perhaps to Six Sigma levels
- *High efficiency* of defect removal activities
- *High reliability* when operating
- *Ease* of learning and ease of use
- *Clarity* of user guides and help materials
- *Ease of access* to customer support
- *Rapid repair*s of reported defects

This is also true for the definition of the ISO 8204 for quality: „Totality of *characteristics* of an entity that bears on its ability to satisfy stated and implied *needs*.“. That means: We require a *quality software product* to have certain characteristics that are related to *requirements* (of the user) and satisfies them. It is clear that the pair *requirement* and *characteristic* plays a central role in the definition of *quality*. Therefore, an object oriented model1 contributes to a better understanding for these notions. Figure.1 shows a software product, which is to fulfill *requirements* in having appropriate *characteristics.* The existence of relationships between *requirements* and *characteristics* makes statements about the quality of a product possible.

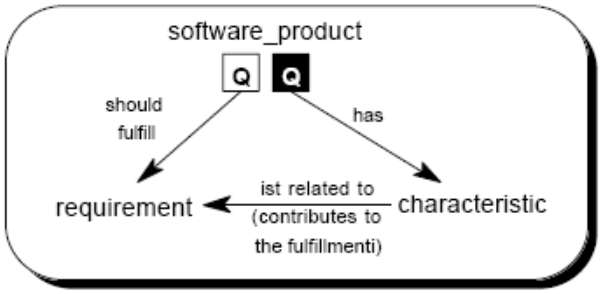

Fig.1 : Relationship between requirements and characteristics in conjunction with quality

Despite the standardized ISO definition of quality, it can be shown that scientific literature lacks consistency and unity regarding the usage of the terms requirement and characteristic. Examples of this heterogeneity are terms like feature, attribute and characteristic.

The following example will show that it is possible to describe the management of requirements and characteristics in the software development process to determine the quality of a software product.

A. Software Quality Factors

There are various factors that affect software quality. In this paper we look at these factors proposed by various people and organizations

i. McCall Quality Factors

- According to this, quality factors can focus on three important aspects of a software product. They are Product Operation, Product Revision and Product Maintenance.
- Correctness: The extent to which a program satisfies its specifications.
- Reliability: The extent to which a program its specifications.
- Efficiency: The amount of computing sources and code required and code required by a program to perform its function.
- Integrity: Extent to which access to software or data by unauthorized persons can be controlled.
- Usability: The ease with which a user is able to navigate to the system.
- Maintainability: Effort required fixing and testing the error.
- Flexibility: Effort required modifying an already operational program.
- Testability: Effort required testing a program so that it performs its intended function.
- Portability: Effort required porting an application from one system to another.
- Reusability: Extent to which a program / sub-program that can be re-used in another applications.
- Interoperability: Extent required to couple one system to another.

ii. FURPS Quality Factors

- Hewlett Packard developed a set of quality factors that has been given the acronym FURPS – Functionality, Usability, Reliability, Performance and Supportability.
- Functionality - is assessed by evaluating the features and capabilities of the delivered program and the overall security of the system.
- Usability - is assessed by considering human factors, overall aesthetics, look and feel and easy of learning.
- Reliability - is assessed by measuring the frequency of failure, accuracy of output, the mean-time-to-failure (MTTF), ability to recover from failure.
- Performance - is assessed by processing speed, response time, resource utilization, throughput and efficiency.
- Supportability - is assessed by the ability to extend the program (extensibility), adaptability, serviceability and maintainability.

iii. ISO 9126 Quality Factors

The ISO 9126 standard identifies six key quality attributes.

- *Functionality* - degree to which software satisfies stated needs.
- *Reliability* - the amount of time the software is up and running.
- *Usability* - the degree to which a software is easy to use.
- *Efficiency* - the degree to which software makes an optimum utilization of the resources.
- *Maintainability* - the ease with which the software can be modified.
- *Portability* - the ease with which a software can be migrated from one environment to the other.

B. Factors affecting Software Quality

Software quality can be seen as having three aspects: functional quality, structural quality, and process quality**.**

i. Functional

Functional quality means that the software correctly performs the tasks it’s intended to do for its users. Among the attributes of functional quality are:

Meeting the specified requirements: Whether they come from the project’s sponsors or the software’s intended users, meeting requirements is the sine qua non of functional quality. In some cases, this might even include compliance with applicable laws and regulations. And since requirements commonly change throughout the development process, achieving this goal requires the development team to understand and implement the correct requirements throughout, not just those initially defined for the project.

Creating software that has few defects : Among these are bugs that reduce the software’s reliability, compromise its security, or limit its functionality.

© Copyright: All rights reserved.

Achieving zero defects is too much to ask for most projects, but users are rarely happy with software they perceive as buggy.

Good enough performance. From a user's point of view, there's no such thing as a good, slow application.

Ease of learning and ease of use : To its users, the software's user interface is the application, and so these attributes of functional quality are most commonly provided by an effective interface and a well-thought-out user workflow. The aesthetics of the interface—how beautiful it is—can also be important, especially in consumer applications.

Software testing commonly focuses on functional quality. All of the characteristics just listed can be tested, at least to some degree, and so a large part of ensuring functional quality boils down to testing.

ii. Technical or Structural quality

The second aspect of software quality, structural quality, means that the code itself is well structured. Unlike functional quality, structural quality is hard to test for (although there are tools to help measure it, as described later). The attributes of this type of quality include:

- *Code testability:* Is the code organized in a way that makes testing easy?
- *Code maintainability*: How easy is it to add new code or change existing code without introducing bugs?
- *Code understandability:* Is the code readable? Is it more complex than it needs to be? These have a large impact on how quickly new developers can begin working with an existing code base.
- *Code efficiency*: Especially in resource-constrained situations, writing efficient code can be critically important.
- *Code security*: Does the software allow common attacks such as buffer overruns and SQL injection? Is it insecure in other ways?

Both functional quality and structural quality are important, and they usually get the lion's share of attention in discussions of software quality.

iii. Process quality

The quality of the development process significantly affects the value received by users, development teams, and sponsors, and so all three groups have a stake in improving this aspect of software quality.

The most obvious attributes of process quality include::

- *Meeting delivery dates*: Was the software delivered on time?
- *Meeting budget*: Was the software delivered for the expected amount of money?

***A repeatable development process that reliably delivers quality software*:** If a process has the first two attributes—software delivered on time and on budget—but so stresses the development team that its best members quit, it isn't a quality process. True process quality means being consistent from one project to the next.

Apart from the above , other factors may include here such as ;*Usage qua*lity, which includes ease of use and ease of learning, *Service quality*, which includes access to support personnel, *Aesthetic quality*, which includes user satisfaction and subjective topics, *Standards quality*, which includes factors from various international standards and *Legal quality*, which includes claims made in lawsuits for poor quality.

III. JOINT APPLICATION DEVELOPMENT (JAD)

JAD (Joint Application Development) is a methodology that involves the client or end user in the design and development of an application, through a succession of collaborative workshops called JAD sessions. Chuck Morris and Tony Crawford, both of IBM, developed JAD in the late 1970s and began teaching the approach through workshops in 1980.

Collecting systems requirements is an inherently difficult process; hence, choosing the correct methodology to accomplish this task is very important. The traditional process consisting of one on one interviewing, questionnaires and observation, is too time consuming due to slow communication and long feedback time. Furthermore, several articles present findings that stress the importance of an effective information gathering process. We outline some industry statistics:

- 60% of large systems projects have significant cost overruns
- 75% of completed systems are in need of constant maintenance
- 60% to 80% of errors originate in the user requirements and functional specifications he JAD philosophy incorporates four simple ideas:
- People who actually do a job have the best understanding of that job.
- People who are trained in IT have the best understanding of the possibilities of that technology.
- Information systems and business processes rarely exist in isolation.
- The best information systems are designed when all of these groups work together on a project as equal partners.

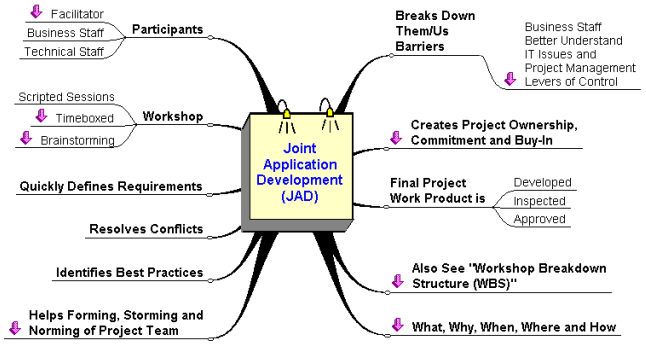

Fig .2 : Joint Application Development Process

The JAD approach, in comparison with the more traditional practice, is thought to lead to faster development times and greater client satisfaction, because the client is involved throughout the development process. In comparison, in the traditional approach to systems development, the developer investigates the system requirements and develops an application, with client input consisting of a series of interviews.

A variation on JAD, rapid application development (RAD) creates an application more quickly through such strategies as using fewer formal methodologies and reusing software components.

A. Advantage of JAD

Several articles and case studies have concluded that the JAD process can yield tremendous benefits. Below is a list that combines the highlights of the findings:

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

- Saves time, eliminates process delays and misunderstandings and improves system quality.
- One of the best ways to reduce scope creep, which results from poor initial requirements.
- Avoids bloated functionality, gold-plating, and helps designers delay their "solution fixation' until they understand the requirements better.
- Lays the foundation for a framework of mutual education, separate brainstorming, binding negotiation, and progress tracking.
- Builds consensus and ownership among users.
- Improves design quality of the deliverable because it forces a definition of that deliverable in advance.
- Project teams get and stay focused.
- JAD forms a natural partnership with modern development tools.

Clearly, the JAD process is extremely powerful and effective. Businesses need to realize the importance of utilizing some form of JAD when embarking on new projects. However, businesses must also realize that no methodology is one hundred percent perfect.

B. Limitations of JAD

The JAD approach encounters the same problems that traditional meetings do. Namely, a limited number of people can participate effectively before the meeting becomes inefficient. Most researchers believe that the maximum effective group size is five, which is much smaller than a typical JAD session. Furthermore, only one person can speak at a time. According to authors Dennis, Hayes and Daniels in 1999, this fact leads to several problems. As one person is talking, other experts are blocked from contributing their ideas and information until it is their turn. During this period, these ideas might be forgotten or suppressed because they seem less relevant or less original at a later time. Basically, vital information may be excluded.

Also, some group members will dominate the topic, while others will remain silent. This domination and inequality of participation and influence can lead to poor-quality models that favor the dominating participants. In particular, this phenomenon will occur in meetings where participants have different levels of status in the company, such as the case with JAD. The normal JAD workshop will include mid-level, as well as, senior managers. Members of upper management usually dominate the meeting and honest participation by subordinates is inhibited.

Another problem that may cause a JAD session to fail is an ineffective facilitator. In fact, many JAD failures can be contributed directly to the facilitator. Furthermore, to add to the concern, research has determined that most leaders and members of organizations are "woefully ill-prepared to meet the challenge of facilitating groups." Basically, businesses need to entrust their projects to capable and dependable leaders.

IV. GROUP SUPPORT SYSTEMS (GSS)

The aforementioned problems can be solved by the implementation of Group Support Systems in the JAD workshop. GSS are integrated computer and communication systems that support group work. Specifically, group members use computers to interact and exchange ideas and information, in addition to discussing topics verbally. The goals of GSS actually address the problems listed in the previous section. GSS attempts to reduce the process loss associated with disorganized activity, member dominance, social pressure, inhibition of expression, and other difficulties associated in the group setting. Furthermore, GSS provides at least three functions that may improve meetings: parallel communication, anonymity and group memory.

First of all, parallel communication allows several participants to contribute information simultaneously. For instance in a two hour meeting with ten participants, group members have twelve minutes to contribute to the topic at hand. The remaining time of 108 minutes is spent not contributing. GSS allows all participants to type ideas simultaneously. Furthermore, the GSS software shares these ideas with everyone.

Second, anonymity may encourage participants, especially lower status participants, to contribute their ideas without the fear of ridicule from their superiors. Anonymity can also help alleviate the pressure of conforming to the ideas of the group. Essentially, anonymity will foster creativity and produce a more fruitful meeting.

Third, group memory is accomplished because all typed information is recorded electronically. All group members will have documented text of all past meetings. This memory may stop problems that occur when information is forgotten or misunderstood. Also, since all members can view the ideas of their fellow participants, they can mold initial ideas with their own, which will, in turn, bolster the creative process.
Group support systems (GSS) are integrated computer and communication systems that support group work.

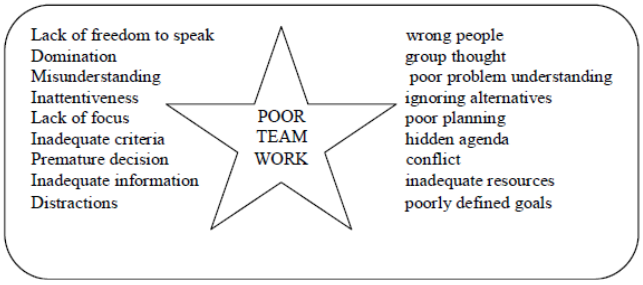

Fig.3: A poor team-work scenario

Other common terms referring to the same concept include, but are not limited to, groupware, group decision support systems, and collaboration technology. GSS facilitate communications, coordination, and decision making by structuring the processes and contents of teamwork. The goals of a GSS are to reduce the process loss associated with disorganized activity, member dominance, social pressure, inhibition of expression, and other difficulties commonly encountered in groups and, at the same time, to increase the efficiency and quality of the end results. Research findings on the impacts of GSS on groups have concluded that the use of a GSS increases task-oriented communication, the quality of decisions, and group members' satisfaction and confidence.

There is an increasing trend toward using computer and communication systems to support such group techniques as brainstorming. Recent developments in GDSS are rooted in earlier efforts to support collaborative work in software development . GSS have been proven useful in such tasks as idea generation, business planning, crisis management, and knowledge acquisition. GSS technologies make group meetings more productive by supporting rapid access to information and message exchanges, by providing better group memory management and feedback, and by facilitating more structured group interaction processes. Groups supported by GSS technologies are able to use more complex group modeling tools to create and explore a wider range of alternatives.

© Copyright: All rights reserved.

In addition, using a GSS often increases participation, reduces meeting time, and improves the quality of meeting results, especially when large groups are involved in complicated tasks.

A. *Limitations of GSS*

The Dennis study encountered three fundamental issues. The first problem was user training. Participants need to be trained on the nuances of the GSS software. The ineffective use of the software will produce undesired results from the process. The authors recommend that participants regularly pause to ask questions and to confirm that they are correctly using the software.

The second problem involved consistency and integration. Essentially, different experts use different terminology, different semantic understandings of the process and different writing styles. Therefore, it may be difficult to incorporate all the information gathered during the meeting into one complete, efficient document.

The final issue was information overload. Traditional meetings can produce volumes of information by themselves. The addition of GSS greatly increases that amount of information. Therefore, participants may have a difficult time assimilating the tremendous amount of information.

V. CASE TOOLS

Computer-aided software engineering (CASE) refers to automated software tools used by systems analysts to develop information systems. These tools can be used to automate or support activities throughout the systems development process with the objective of increasing productivity and improving the overall quality of systems. CASE tools can be classified as either upper, middle and lower. With regards to systems analysis, Upper CASE tools are primarily used because they "support models of enterprises, business environments, and planning.". Furthermore, they are used in the application of various structured techniques including data modeling using entity relationship diagrams, process modeling using data flow diagrams, and structured design using structure charts.

Computer-Aided Software Engineering (CASE) was developed to support software projects and improve many aspects of systems development, CASE itself is an encompassing term for an extremely wide variety of tools and methods that automate or aid designing, documenting, and producing structured computer code tasks, testing, maintenance and even for upgradation of an existing software products. By providing this high-level of support to developers they themselves no longer needed to perform as much background work such as code checking or manual modeling redesigns meaning more time and resources to focus on the software system itself.

These tools and methods can be classed into these general categories -

- Data Modelling
- Model and Program Transformation
- Refactoring
- Source Code Generation
- Unified Modelling Language
- Designing test cases
- Documentation

A. *Data Modeling*

Data modeling is a method used by developers to define data requirements as data models representing the business processes of the desired system. These data models define the relationships between data elements and structures in a standard manner and support system design by providing the definition and format of data.

Data models should be considered progressive, as usually there's no definite final data model for a system. Instead data models should be considered living documents that change in response to development progress, changing requirements, and other variables; which means the documents reflect the reality of the system much more accurately. Data models ideally should be stored in a central repository so that they can be retrieved, expanded, and edited over time as well as making it easy to supply system information to all members of a development team.

There are three stages of Data Models –*Conceptual, Logical and Physical*

Conceptual data models are the first step of data modeling, they represent an abstract implementation of the system in terms of the semantics of requirements, significant system entities and the assertions that link those entities together. This abstract mode has no specific implementation forms; they exist to provide a view of the system from its requirements showing the final pattern in a more visual form.

Logical data models take the conceptual models of the specific problem domain in question and implements it in terms of a particular data management technology but without being specific to any particular language. By defining the model in terms of a technology such as relational tables or object-oriented classes you provide the base needed to create the model in the physical world through an implementation of the technology chosen, with all the design work done by this point each part should be able to be constructed with little to no problems

Physical data models take the logical model and assign attributes to each object, variable, and method (if used) entity that contains details of their specific real-world implementations. For example a variable is an integer with length five provides the implementation attributes for that variable usable in a specific implementation language or software such as SQL.

CASE Data modeling means adding layers of design onto requirements but within logical boundaries; for example a human designer might make an error and create a flawed model, a computer can detect such errors and either correct or point them out. Using computer assisted data modeling allows much more error-free versions of the design to be created with less risk.

B. *Model and Program Transformation*

Both of these transformation types are based on the same core principles, they take in a source object of one meta-model and produces a transformed target object of a different meta-model; a simple example is converting a text document to a word document. Having automated tools to perform these action guarantees the resulting output is correct because the tools are programmed to create specific output from input, the same output would be created every time that particular input is read.

Model-Driven Engineering uses the concept of Model Transformation to allow domain-less concepts to occur; by domain-less I mean a system that has no implementation domain such as Windows or Mac. Model transformation provides the concepts necessary to allow a system to be engineered and then transformed into various suitable forms while still retaining the original model.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

C. Refactoring

Code refactoring is the process of modifying a systems internal structure without changing the external behavior or functionality, an easier way to see this is improving source code without changing the overall results from that code. The reasons for refactoring generally fall under a desire to modify code to fit a specific coding style and to improve various aspects of the code such as readability, performance, coding structure, or other quality attributes.

Refactoring can be seen as a special form of Program Transformation as it takes a source such as a code variable and transforms it throughout the system into an output. These transformations are intended to make the code easier to understand, less resistant to change and more maintainable.

Like transformations refactoring a program is going to have better results when done by a machine as the machines never going to miss a variable name as it appears throughout code while a human could. Being able to refractor sections of code on the fly makes it easy to modify code to fit the desired mold, which in turn makes it easier to implement that mold.

D. Source Code Generation

This is a method for generating code based on using an ontological model of the source language containing a domain of language concepts and the relationships between those concepts. This model allows a program such as an Integrated Development Environment (IDE) or template parser to decide what code should be added to a system without needing to get human approval.

Source Code Generation is used in a variety of highly useful ways ranging from generating source code in the background for a user created graphical interface to suggesting and filling in small snippets of code while a programmer is programming; for example auto-filling a variable name from the start of a word is a small use of source code generation. It's a useful method that removes the need for a programmer to deal with lots of small jobs through automation and can improve software quality a great deal if the results are correct.

This CASE method also covers one of the most important aspects of programming, Compiling. The act of compiling a program to a different usually lower-level language is one that could only be done by machines except in the very smallest of programs. Taking the program as the source and with a model of both the source and target languages this method is able to translate all the code into its equivalent in the target language without losing functionality or program aspects. Being able to compile code automatically makes it fast, efficient, and easy to change the original program without being responsible for retranslating it each time.

E. Unified Modeling Language (UML)

UML is a standardized general-purpose visualizing modeling language used to construct and document objects and object-oriented systems, equivalent to creating a blueprint for a software system. UML is usable throughout the Software Development Life Cycle and combines multiple-methods of modeling including workflows, object, component, and entity relationship modeling techniques.

UML diagrams form the core of the language and come in two general forms –

- Structural Form – Takes static aspects of the system such as objects, attributes, methods and the relationships between to form a structure representing the system.
- Behavior Form – Captures the dynamic behavior of the system as it would unfold if the system was running using behavior-centered diagrams including activity, sequence, communication and state machines.

UML programs as well allow whole designs to be transferred between developers and the design can contain much detail and depth which provides a fuller and more accurate model of the system to the designers and developers, as a result end-productions can be of a higher quality than with for example a group of paper designs. An extra bonus is that the quality of the actual diagrams themselves is much sharper leaving less room for misunderstandings about the designs.

UML is widely used throughout computing as a useful tool for aiding the development of a software system through its ability to quickly bring a system together with visual diagrams depicting its form. A UML program allows system elements to be designed quickly and modified easily giving much more flexibility to the developers to spend more time on higher-quality solutions for the system.

F. Design Test Cases

A series of testing processes are conducted to provide software product with better quality. We include few testing methods below.

Find defects. This is the classic objective of testing. A test is run in order to trigger failures that expose defects. Generally, we look for defects in all interesting parts of the product.

Maximize bug count. The distinction between this and "find defects" is that total number of bugs is more important than coverage. We might focus narrowly, on only a few high-risk features, if this is the way to find the most bugs in the time available.

Block premature product releases. This tester stops premature shipment by finding bugs so serious that no one would ship the product until they are fixed. For every release decision meeting, the tester's goal is to have new showstopper bugs.

Help managers make ship / no-ship decisions. Managers are typically concerned with risk in the field. They want to know about coverage (maybe not the simplistic code coverage statistics, but some indicators of how much of the product has been addressed and how much is left), and how important the known problems are. Problems that appear significant on paper but will not lead to customer dissatisfaction are probably not relevant to the ship decision.

Assess conformance to specification. Any claim made in the specification is checked. Program characteristics not addressed in the specification are not (as part of *this* objective) checked.

Assess quality. This is a tricky objective because quality is multi-dimensional. The nature of quality depends on the nature of the product. For example, a computer game that is rock solid but not entertaining is a lousy game. To *assess* quality -- *to measure and report back* on the level of quality -- we probably need a clear definition of the most important quality criteria for this product.,

Verify correctness of the product. It is impossible to do this by testing. We can prove that the product is not correct or can be demonstrated that we didn't find any errors in a given period of time using a given testing strategy. However, we can't test exhaustively, and the product might fail under conditions that we did not test. The best we can do (if you have a solid, credible model) is assessment--test-based estimation of the probability of errors.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

Assure quality. Despite the common title, quality assurance, we can't assure quality by testing. We can't assure quality by gathering metrics. We can't assure quality by setting standards. Quality assurance involves building a high quality product and for that, we need skilled people throughout development who have time and motivation and an appropriate balance of direction and creative freedom.

G. Documentation

The documentation of a program is very important for software development, the functional specification for example is a core document that all other software development will be based on; tools that can improve the success of such a document can lower future risks to the software system development.

VI. INTEGRATION OF JAD, GSS AND CASE TOOLS

In terms of the JAD session, CASE tools are increasingly being used. Systems developers are better able to work closely with users in defining system requirements when it is possible to show them CASE tools-generated graphical models. Moreover, the article stipulates that CASE tools can help with the employment of GSS in the JAD sessions. Systems specifications, screen designs, and flow charts generated during the GSS-supported JAD sessions can be used as inputs to CASE tools for formal systems specifications, code generation and documentation. In conclusion, the introduction of CASE tools during the JAD requirement gathering stage of project development has effectively enhanced the design stage of the process.

Research results indicating positive outcomes of using GSS for systems development activities have made it appear that an ideal approach would be to employ the JAD methodology with appropriate GS S tools to support requirements elicitation. Therefore, equal participation may be achieved, group consensus may be assessed in real time, and semiformal specifications may be generated in group meetings and recorded electronically . Moreover, results from requirements elicitation sessions can be exported and transformed into specific formats supported by existing CASE tools to speed up the systems development process.

Many GSS applications have consistently fallen short of expectations, and conflicting reports of the usefulness of GSS to facilitate various group activities call attention to the importance of adopting a methodological approach to using GSS. The success of GSS may depend on the nature of the application domains in which they are introduced and the methodologies employed in using them. The existence of a methodology for guiding the use of GSS possibly could have prevented earlier possible misuse and abuse of GSS. Based on our experiences in using GSS in knowledge elicitation, systems analysis and design, and business planning, we have developed a domain analysis methodology (DAM), which is independent of any specific GSS, to support requirement elicitation using GSS, JAD, and CASE tools.

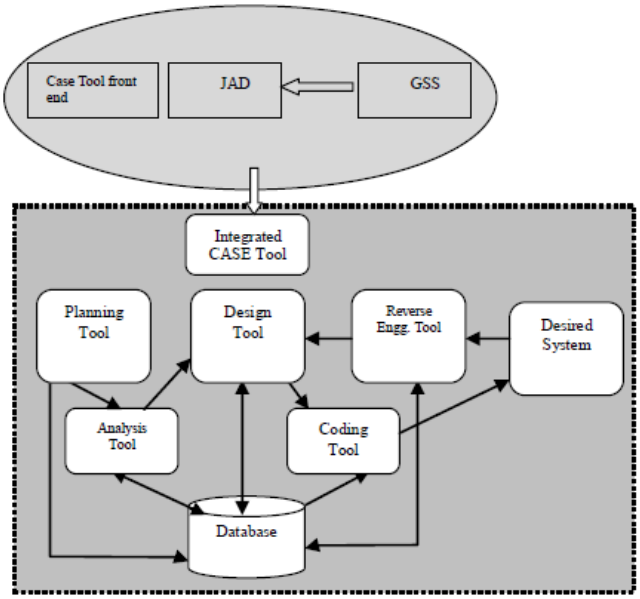

Fig. 4 : Integration of JAD, GSS and CASE Tool

VII. CONCLUSION

Most of the cases Software quality is influenced by inappropriate involvement of users and managers, resulting in incomplete or inconsistent requirements. Therefore, there is a growing interest in involving users in the systems development process, particularly in the planning and analysis phases where conceptual data and process models of an application or an enterprise wide information system are defined. Businesses are also interested in understanding their business processes and reengineering these processes in order to achieve dramatic quality improvement. Business engineering requires full participation of managers and reengineered business process models are the foundations of information systems developed to support them. We have proposed a framework to integrate GSS, JAD, and CASE to support requirements specification. Such an approach can reduce the need for a highly skillful JAD session leader, a kind of specialist who is always in short supply. The framework would also increase users' participation and satisfaction in the process.

Results of a pilot study have been presented. Further analysis and follow-up studies are underway to validate the effectiveness of the proposed approach and using empirical results to refine our framework. We are also planning to build software bridges based on CASE Data Interchange Format standards to integrate groupware products and CASE products.

The contribution of this research is twofold. One benefit pertains to our understanding of activities and processes in software development, how they may be facilitated by a GSS, how they may be structured to achieve the most effectiveness and efficiency, what process models may be employed, what features of a GSS are most useful, and how GSS and JAD are integrated into the domain analysis methodology. The other pertains to the use of GSS for requirements specification in practice and thus to the improvement of applications development productivity in an efficient and effective way

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

REFERENCES

[1] ANSI/IEEE Std 830-1984. IEEE Guide to Software Requirements Specifications. New York: Institute of Electrical and Electronics Engineers, 1984.

[2] August, J.H. Joint Application Design. The Group Session Approach to System Design. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[3] Boehm, B.W. Software engineering. IEEE Transactions on Computer (December 1976), 225-240.

[4] Brennan, P.F. CASE tools in a JAD workshop. CASE Trend, 2, 2 (1990), 5-6, 8.

[5] Brooks, F.P. The Mythical Man-Month: Essays on Software Engineering. Reading, MA: Addison-Wesley, 1975.

[6] Carmel, E.; Whitaker, R.D.; and George, J.F. PD and joint application design: a transatlantic comparison. Communications of the ACM, 36, 4 (June 1993), 40-48.

[7] CDIF--Standardized CASE Interchange Meta-Model. EIA Interim Standard, EIA/IS-82. Washington, DC: Electronics Industries Association, July 1991.

[8] Chen, M.; Liou, Y.I.; and Weber, E.S. Developing intelligent organizations: a context-based approach to individual and organizational effectiveness. Organizational Computing, 2, 2 (1992), 181-202.

[9] Chen, M., and Nunamaker, J.F., Jr. The architecture and design of a collaborative environment for systems definition. Data Base, 22, 1/2 (1991), 22-29.

[10] Chen, M.; Nunamaker, J.F., Jr.; and Weber, E.S. Computer-aided software engineering: present status and future directions. Data Base, 20, 1 (1989), 9-13.

[11] Cook, P., et al. Project Nick: meeting argumentation and analysis. ACM Transactions on Office Information Systems, 5, 2 (1987), 132-146.

[12] Couger, J.D. Evolution of system development techniques. In J.D. Couger, M.A. Colter, and R.W. Knapp (eds.), Advanced System Development/Feasibility Techniques. New York: Wiley, 1983, 6-13.

[13] KEN LUNN – 1.SOFTWARE DEVELOPMENT WITH UML (2002)

[14] .Michael J. Pont - Software engineering with C++ and CASE tools (1996)

[15] Wikipedia on CASE http://en.wikipedia.org/ wiki/ Computer-aided_software_engineering

[16] Wiki on various CASE methods with a focus on Transformations http://www.program-transformation.org/Transform/WebHome

Dillip Kumar Mahapatra has completed his maser degree in CSE and having more than seven years in teaching UG and PG levels. He has published 15 papers in different journals of national level. He has also authored ten text books in the field of CSE and Information technology

Tanmaya Kumar Das has completed his master degree in CSE and having 22years experience in the field of teaching and industries and having more than 19 papers published in journals of national levels. He has also authored more than 12 of books in the field of engineering for UG and PG students.

Prof.(Dr.) Gopa Krishna Pradhan has been awared with Ph.D degree in Computer Science and Engg. from IIT, Kanpur and Ex- Professor in the department of Computer Sc. and Engg. SOA University. He having 40 years of teaching experience in the field of CSE and IT.

© Copyright: All rights reserved.