INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY (IJCET)

ISSN 0976 – 6367(Print)
ISSN 0976 – 6375(Online)
Volume 4, Issue 3, May-June (2013), pp. 389-409
© IAEME: www.iaeme.com/ijcet.asp
Journal Impact Factor (2013): 6.1302 (Calculated by GISI)
www.jifactor.com

IJCET
©IAEME

SELECTION CRITERION AND IMPLEMENTATION OF CASE TOOLS IN GAP ANALYSIS TOWARDS DISTRIBUTED SOFTWARE DEVELOPMENT

Dillip Kumar Mahapatra[1], Gopa Krishna Pradhan[2]

[1](Head of the Department of Information Technology, Krupajal Engineering College/Biju Patnaik University of Technology, Rourkela, Odisha, India)
[2](Ex. Professor, Department of Computer Science & Engineering, SOA University, Bhubaneswar, Odisha, India)

ABSTRACT

Software is growing ever-more complex and new software processes, methods and products put greater demands on software engineers than ever before. Software Engineering is broadly associated with the development of quality software with increasing use of software preparation standards and guidelines.

Organizations now have a tendency to make greater efforts in developing software in distributive way. The main advantage of this lies in a greater availability of human resources in decentralized zones at less cost. There are, however, some disadvantages which are caused by the distance that separates the development teams. Coordination and communication among the team members become more difficult as the software components are sourced from different places, thus affecting project organization, project control, and product quality. In addition to these, there are major challenges like technical diversities such as hardware and software configuration of distributed site, product architecture, development methodology, managerial techniques result with "gaps" at different levels of distributed software development project. There are different methodologies to deal with these gaps.

The use of Computer Aided Software Engineering (CASE) tools has been marketed as a remedy for the software development crisis by automating analysis, design, and coding for the ease the distributed development of software systems.

This paper proposes the selection of appropriate CASE tools and their implementation in analysis gaps towards managing distributed software development.

Keywords: Distributed software development, GAP, Integrated CASE Tools, Selection, Implementation, Cross-cultural issues/cultural differences, coordination theory, interpretive methods, software project management, virtual teams/geographically dispersed teams.

I. INTRODUCTION

Globally distributed software development achieves division of labor by dispersing software development tasks among several remote development centers. This mode of software development has become a popular business model for software organizations(Ref.01).

There are several compelling business reasons supporting the adoption of distributed software development such as ; ability to extend work beyond the regular office hours at a single site, software development costs at offshore centers, like in India, are as much as four times less expensive, the capabilities of workforce in remote centers located in emerging economies have improved significantly in the recent years, advances in information and communication technology have facilitated easier collaboration between remote workforce.
At the same time several gaps in distributed software development have been reported. For example, distributing product maintenance work across global development centers increases the cycle time of a project, cross-site work takes a much longer time and requires more effort for work of similar size and complexity. Also behavioral researchers investigating distributed work report that a remote workforce, even with advanced technologies in place, often encounter difficulties in coordination and administration that lead to decreased project performance. Structured and disciplined software engineering processes have often been advocated as a key remedy for addressing the aforementioned challenges. In addition to this, CASE tools play important role to analyze these gaps to a better extent.

In this paper, we prioritize the selection and implementation of CASE tools to analyze gaps in distributed software development.

II. GAPS IN DSD

The term “GAP” can be viewed as the situation that influences the performance of a system i.e the actual performance of the system is different from its expected performance. From distributed software development point of view, there are number of influencing factors that affects the successfulness of the project from managerial, technical and the most importantly the user’s point of view(Ref. 03).

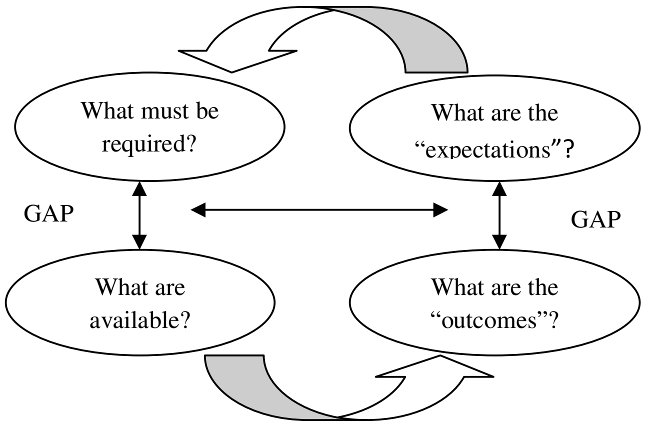

Fig.1: GAP (a general view)

A list of dimensions to conceptualize gaps in DSD is;

- Geographical distance
- Time zone differences
- Governance differences
- Inadequate communication
- Socio-cultural differences
- Knowledge management
- Project and process management
  - S/w architectural difference
  - Technical and infrastructural differences.

2.1 GAP ANALYSIS

In general point of view, *Gap Analysis* is the process through which a company compares its actual performance to its expected performance to determine whether it is meeting expectations and using its resources effectively. Gap analysis seeks to answer the questions "where are we?" (Current state) and "where do we want to be?" (Target state).

Gap analysis provides a foundation for measuring investment of time, money and human resources required to achieve a particular outcome. 'GAP analysis' has also been used as a means of classifying how well a product or solution meets a targeted need or set of requirements(Ref.05).

Conducting a gap analysis can help a company re-examine its goals to determine whether it is on the right path to be able to accomplish them. A company will list the factors that define its current state, outline the factors that are required to reach the target state, and then determine how to fill the "gaps" between the two states.

In software development, for instance, a gap analysis can be used to document which services and/or functions have been accidentally left out, which ones have been deliberately eliminated and which still need to be developed. In compliance, a gap analysis can be used to compare what is required by law to what is currently being done.

As previously mentioned, Distributed Software Development (DSD) Project is conducted at the different locations of the globe; gaps are to be analyzed to take precautionary measure to maintain the quality, integrity of the project and its outcomes from the developers and users prospective.

The success of a DSD projects gap analysis depends on proper preparation, research and consideration of all internal and external factors. Organizations conduct gap analysis to identify the limitations of DSD or the differences between current and desired levels of carrying out development.

2.2.1 APPROACHES OF GAP ANALYSIS

Performing a gap analysis is a great way to determine what the next course of action for a project or process improvement undertaking should be. Gap analysis is a tool that project managers, process improvement teams, and even individuals use to see where a person or company is when compared with where that person or company would like to be. There are various gap analysis methods that can be undertaken when utilizing this tool in DSD projects. Read on to learn about different gap analysis methods(Ref.08).

Traditional Gap Analysis

Gap analysis relies upon taking a critical look at what the current standing or situation is for a person to a company. In order to make any improvements in a company, the first step is to understand where we are and where we want to be. It can be understood, where you want to be by looking at the company's mission statement, strategic objectives, and improvement goals. The first step, then, in performing a gap analysis is to define where we want to go in as specific terms as possible.

Once it is known where we're headed, we can collect data on where we are currently. The second step in performing a traditional gap analysis is to look at things as they are in the company.

After putting together the "where we want to be" against the "where we are," we can start to see the different steps that will need to be taken in order to bring about the desired effects.

SWOT ANALYSIS

An SWOT analysis is another gap analysis method that can help to balance where the project is against where it should to be. SWOT actually stands for "Strengths," "Weaknesses," "Opportunities," and "Threats." When undertaking an SWOT analysis, the first step is to list all of the project's or department's strengths. What makes the department so great? What is working for the project or the project team?

Next, a list to be formed, including all the weaknesses of the projects, departments or companies. Take a look at the ways in which things are not going so well. What could be done better?

Third, list opportunities that exists. What areas are there that the company, project, or department could take hold of? Are these opportunities realistic?

Fourth, list any threats that are presented that may derail the company, department, or project. Is there anything that can be done to prevent these threats?

Finally, the aim will be to maximize strengths and take advantage of opportunities that present themselves while minimizing company weaknesses and avoiding threats.

HOW SDLC IS AFFECTED BY GAP?

Software Development Life Cycle (SDLC) highlights a set of phases to carry out the projects in distributive manner is greatly affected by the gaps as discussed below(Ref. 11):

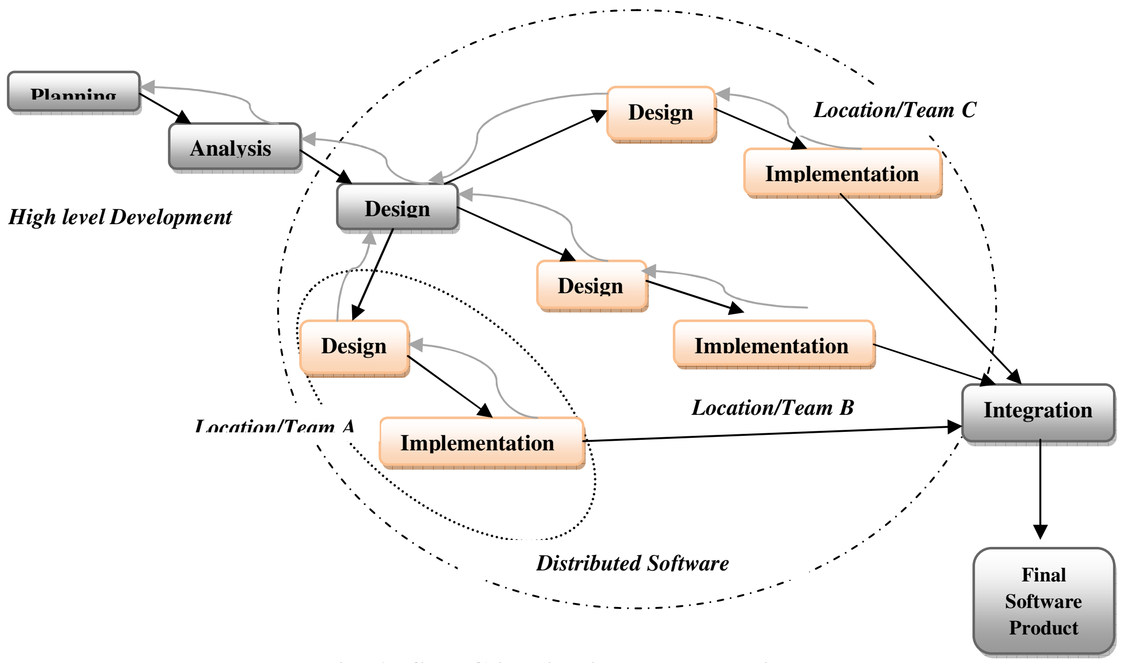

Fig. 2: SDLC in distributed scenario

Planning phase: This phase includes

- Requirements Identification
- Feasibility Analysis
- Project management and results with *Project work place*

Analysis Phase: This includes

- Develop analysis strategy
- Define requirements
- Use cases, Process models and data models and outputs *system proposal*

Design Phase: This includes

- Design selection
- Architectural design
- Interface design
- Data storage design
- Program design and results with *system specifications*

Implementation Phase: This includes

- System construction
- Installation Process
- Support Plan and produces efficient, robust and functional system (product)

Mostly, these projects are large and complex and conducted in iterative incremental way. A clear co-ordination among the teams at different locations w.r.t. different organizations is required. A continuous management policy must be injected during the project. The development processes are affected by mix of skill, knowledge and expertise at multiple sites. Distributive Risk management is also a great influencing factor and Time, language and cultural diversities among sites affect the timely completion of the project. Software Configuration management is a major issue in distributed software development. Integration of systems from different locations is a great challenge and even migration of software (legacy) to another environment is also a constraint. Overall all the phases of distributed software development process are affected by the gaps.

2.2.2 FOUR KEY STEPS IN PERFORMING A GAP ANALYSIS

Another method of undertaking a gap analysis is through seeing the analysis process as four key steps:

1. Seek an understanding of the environment surrounding the problem or project.
2. Take a wholistic view of the environment to gain a complete understanding.
3. Determine what framework your team will use for assessment of the problem or project
4. Make sure to provide data supporting the analysis have undertaken.

By keeping these four steps in mind while undergoing a gap analysis, it can be insured that the results of the analysis are accurate and useful.

III. UTILIZING CASE TOOLS FOR ANALYZING GAP

CASE tools are the software engineering tools that permit collaborative software development and maintenance. CASE tools support almost all the phases of the software development life cycle such as analysis, design, etc., including umbrella activities such as project management, configuration management etc(Ref.06). The CASE tools in general, support standard software development methods such as Jackson Structure programming or structured system analysis and design method. The CASE tools follow a typical process for the development of the system, for example, for developing data base application, CASE tools may support the following development steps:

- Creation of data flow and entity models
- Establishing a relationship between requirements and models
- Development of top-level design
- Development of functional and process description
- Development of test cases.

The CASE tools on the basis of the above specifications can help in automatically generating data base tables, forms and reports, and user documentation.
Thus, the CASE tools –

- support contemporary development of software systems, they may improve the quality of the software
- help in automating the software development life cycles by use of certain standard methods
- create an organization wide environment that minimizes repetitive work
- help developers to concentrate more on top level and more creative problem-solving tasks
- support and improve the quality of documentation (Completeness and non- ambiguity), testing process (provides automated checking), project management and software maintenance.

Because of the nature of the distributed software development, there are different types of CASE tools and the role of tools within the context of the software development process, the methods used within an organization, the hardware and software environment, and the personnel who will use and maintain the system. A number of technical issues including tool integration, data management, performance, maintainability, and standardization are there. In addition, non-technical issues of relevance to managers are considered including the tool selection, the adoption process, and culture change. Each of these issues represents an important consideration in the formulation of a CASE strategy.

3.1 IMPACT OF CASE TOOLS

It is imprecise to make blanket statements about the benefits of CASE technology due to the diverse nature of CASE tools. Some tools, such as configuration management tools and documentation tools, are generally accepted mechanisms for improving the manner in which software is developed(Ref. 12). More controversial are the benefits of other tools, such as the many analysis and design tools, reverse engineering, or code generation tools which are commercially available. Among the major problems with careful measurement of the impact of any type of CASE tool are:

- The wide variation in quality and value within a single type of tool.
- The relatively short time that many types of CASE tools have been in use in organizations.
- The wide difference in the adoption practices of various organizations.

• The general lack of detailed metric data for previous and current projects.
• The wide range of project domains.
• The confounding impact of changes to methods and processes that is often associated with the adoption of CASE tools.
• The potential bias of organizations reporting CASE gains or losses.

Illustrative of the problem with measuring CASE impact are the varying experiences of organizations that have purchased and used CASE analysis and design tools.

The most consistent benefits cited in the CASE literature are improved communications and documentation. Communications appears to be enhanced both between the project engineers and the customer and among engineers working on a project. The improvement is often attributed to more accurate, consistent, and understandable representations of a system which are potentially possible with CASE tools. The experienced improvement in documentation relates to this greater consistency and accuracy for specifying the system and to the direct generation of portions of the documentation by the CASE tools.

Some necessary features that must be supported by CASE tools in addition to the above are:
• It should have Security of information. The information may be visible/ changeable by authorized users only.
• Version Control for various products.
• A utility to Import/Export information from various external resources in a compatible fashion.
• The process of Backup and Recovery as it contains very precious data.

3.2 MAJOR CASE ISSUES

Clearly, the decision to invest in CASE technology is not easy to make. An organization must make informed assessments about the value of a wide variety of individual tools based on inconclusive data. Determining the potential value of a tool is made even more difficult to measure in isolation, since an increasing number of CASE researchers and users have reached the conclusion that the greatest value of CASE tools will only be achieved when used in concert with other tools (i.e., integrated)(Ref. 12).

Organizations that are attempting to reach consensus on the purchase of CASE tools must address a number of issues. The positions adopted by an organization can determine whether a CASE tool will be ultimately successful. Among the more vexing issues are:

• **Investment of Resources**. The cost of adopting CASE technology.
• **Current Processes and Methods**. The frequently inexact match between the processes and methods supported by CASE tools and those utilized within the organization.
• **Support Mechanisms.** The extensive support systems necessary for CASE tools.
• **Tool Scalability.** The somewhat limited capabilities of CASE tools to deal with very large systems.
• **Assessment of Real Value.** The difficulty in determining the actual value of CASE tools when faced with sometimes inflated claims from vendors.
• **Standards Selection.** Selecting among the competing and conflicting standards, supported by CASE tools.
• **Adoption Complexity.** The complexity of the tool adoption process.

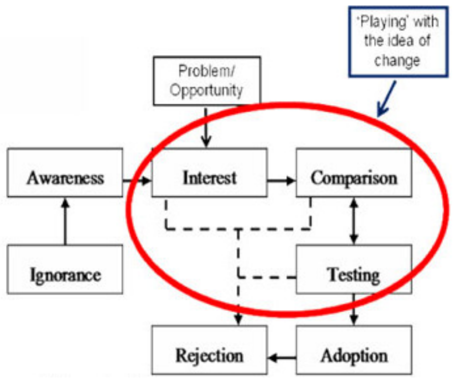

Fig.3: CASE Adoption Stages

3.3 THE CASE TOOL DOMAIN

The CASE tool domain contains a variety of products and features which are dependent on the application area, the hardware platform, the operating system and the user interface required. The tools cover the planning, analysis, design and construction stages. The way in which these stages are implemented depends on whether there is a method associated with their use or whether the techniques are generic. The ability to import and export data to other tools and to retrofit existing systems are also features which can exist. The power of the tool depends on the level of sophistication of the central repository of data and this can vary from a simple data dictionary to a knowledge based encyclopedia(Ref.03).

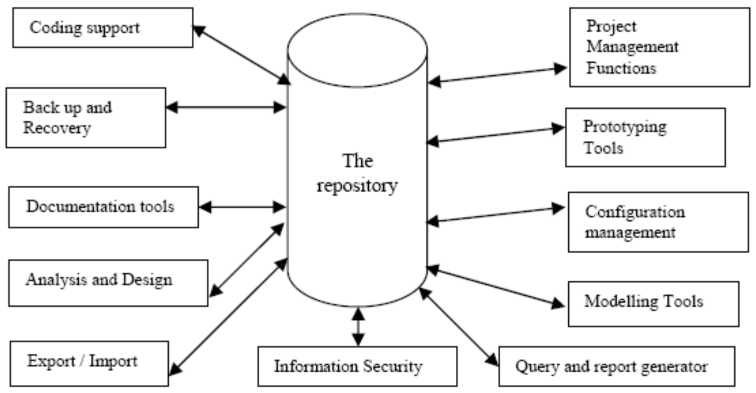

Fig 4: A CASE environment

These features are discussed below:

Application Area: This may vary from commercial systems which might be batch or transaction oriented system to real-time systems which may involve process control or embedded systems.
Hardware: Many of the tools allow development on an intelligent workstation which allows developers to work on individual parts of the project. The results of the analysis and design can then be exported to the target hardware for construction and implementation.

Operating System: This largely divides the products into those that can be used with UNIX and those can run on DOS or OS/2 with exporting to IBM or DEC environments. There is movement toward products being available on a variety of platforms. The networking environment will determine whether teams have multi-access to the data and the tool will determine how effectively the data can be shared while maintaining integrity.

User Interface: This will determine whether how effectively a developer can work with different levels of the tool at the same time.

Integration: The tool can exist as a fully integrated tool covering all phases of the life cycle or it may only perform analysis and design. If this is the case then there may be the need to integrate it with other tools. The ability to import and export the data can be important issue.

Repository: This is the heart of the tool and acts as a meta-database to store information about the functions, procedures, processes, data models, process models, entities and their relationships and so on. The data should allow the user to view it in a variety of ways such as data flow diagrams, structure charts, in a consistent manner with appropriate rules being applied. The repository can exist at different levels of sophistication. In a powerful repository all information is stored, without redundancy, in only one place. If it is distributed then it should also be able to be consolidated into a master repository. A primitive repository would enforces that the user has to provide the link between the diagram components and the data dictionary manually. This can result in inconsistencies which take time to resolve.

Workbenches: Workbenches are the tools which most people associate with the market terminology of CASE. There are a range of tools which are an important part of system development and these will be discussed later.

The workbench tools can be subdivided further according to the level of lifecycle support.

(i) *Planning* - association matrices, organizational hierarchy, subject area, presentation graphs, function hierarchy, function dependency, document graph;

(ii) *Analysis* - entity relationship models, entity life history, data model diagrams, data flow, process hierarchy, process dependency, process action, structure chart, control specification, state transition, matrix processor, prototyping;

(iii) *Design* - dialog flow, screen design, report design, prototyping, pseudo code developer, procedure action diagrams, data structure, database design, database construction; and

(iv) *Construction* - code generation for COBOL, C, Ada, DB2 etc, test data generation, performance optimization, configuration management.

On the basis of their activities, sometimes CASE tools are classified into the following categories:

1. Upper CASE tools
2. Lower CASE tools
3. Integrated CASE tools.

Upper CASE: Upper CASE tools mainly focus on the analysis and design phases of software development. They include tools for analysis modeling, reports and forms generation.

Lower CASE: Lower CASE tools support implementation of system development. They include tools for coding, configuration management, etc.

Integrated CASE Tools: Integrated CASE tools help in providing linkages between the lower and upper CASE tools. Thus creating a cohesive environment for software development where programming by lower CASE tools may automatically be generated for the design that has been developed in an upper CASE tool.

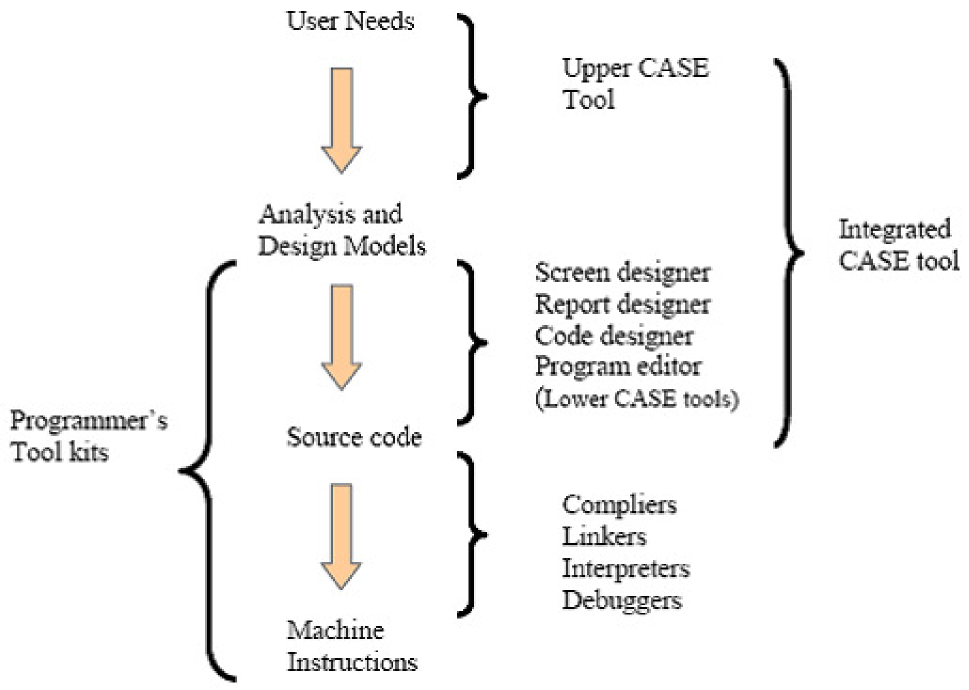

Fig.5: The CASE tools and application development

Application Generation: The 4GL and relational database environment provides a sophisticated development environment where applications can be generated and prototyping is supported. These tools in principle include data modeling, specification facilities and efficient performance.

Retrofitting: This is a term which encompasses reverse engineering and re-engineering. Reverse engineering takes an existing system and recovers the original design specification from the current physical implementation. Re-engineering performs reverse engineering, modifies the existing system and then creates a new improved system(Ref. 07).

IPSE: This stands for Integrated Project Support Environments and is a group of tools which provides support for project management, configuration management, version control, requirements tracing, quality assurance.

3.4 CASE TOOL SELECTION PROCESS

A systematic selection process was generally adopted which involved(Ref. 08):

(a) Establishing the functional requirements (desirable and necessary) of the CASE software required.

(b) Obtaining literature and information from vendors.

(c) Using checklists to review the packages against the requirements covering general, technical, installation and operation, modification and maintenance considerations, vendor capabilities and personnel training. General software package requirements must also be considered. The use of metrics can also be a useful procedure.

(d) Evaluating the products against the desirable requirements that have been established.

(e) Selecting products for demonstrations, vendor visits and hands-on trial and

(f) Making a recommendation based on factors including cost,learning curve,vendor.

3.5 DESIRABLE FEATURES OF CASE SOFTWARE PRODUCTS

To assist us in assessing the products we established a list of desirable features and queries relevant to these features. This checklist along with our requirements and goals provided the basis for selection. The following features, obtained from a variety of sources are considered to be generally the most important in CASE products(Ref. 13):

(a) Support for the entire application development life cycle;
(b) Use of an information repository;
(c) Support for diagramming techniques;
(d) Availability of the product for the existing hardware platform;
(e) Open software architecture;
(f) Compatibility with current software - DBMS, application generators;
(g) Support for structured methodologies;
(h) Support for prototyping tools; and
(i) Use of a well-defined life cycle process.

3.6 IMPLEMENTATION DOMAIN OF CASE TOOLS

Important queries were drawn from the literature and our own requirements analysis are outlined below.
Life Cycle Coverage - What parts of the life cycle is the tool applicable for?
Method - Is the tool based on a recognized method (for example, information engineering, structured analysis and design)?
Techniques - Do the diagramming editors support various techniques? For example in data flow diagrams, are the Yourdon/De Marco and Gane/Sarson techniques supported? Is the Jackson technique supported in structure charts? What is the technique used for entity relationship modeling? Is there consistency checking and balancing?
Integration - Does the tool move from one function to another consistently with a seamless appearance. A complete integration would involve moving smoothly through front-end CASE design to a forms/4GL applications builder, database, network software and query and report generators.
Repository - What is the level of sophistication of the data dictionary? (The level will determine if specifications can be shared across functions). Is the repository in a format which can be accessed by a DBMS? Does the structure of the repository allow data to be accessed by the various tools? How is the repository populated with meta-data regarding the data and processes?
Is the repository held on individual workstations, centrally or in distributed fashion?
Functionality - Does the software should provide error detection, debugging facilities, data integrity, backup and recovery, documentation and auditablity.
Planning - Does the tool provide the means to model the organization?
Analysis - What type of analytical tools are available? Is consistency-checking within the data flow diagrams performed immediately or on request? Are there rules associated with various diagrams? How are violations to the methods reported?
Design - What type of design phase tools are available? Can the tool generate physical design specifications from logical design specifications? Are design specification reports generated automatically? Do the design tools allow diagrams to be exploded to many lower and more specific levels?

Code Generation - Does the tool-set generate code from the graphical specifications? Can it generate code suitable for a variety of hardware? What is the extent of this code: Is it processing logic, data schema, a/or input/output code? If schema is produced, is it normalized?
Documentation - Does the tool produce documentation for the developer? Is there integration with a word processor? Can "live" links be established between the tool and the word processor?
Forms and Graphics - Does the tool provide a screen designer for menu dialogues, screen panels? Do the forms have full repository support? Are multiple windows supported?
Are pop-up menus supported? Is there forms navigation logic through menus?
Query and Reporting - Is the query language easy to use so that the developer can design a simple query as well as a complex query across several tables? Is the report write integrated to the repository? Can the report data be downloaded to decision support software? Do report formats exist as templates? Can templates be easily constructed?
Application Maintenance - What is maintenance performed on, the design specifications or the Code? What is the impact of altering the source code on the repository? Can existing applications be reverse-engineered? Can existing applications be re-engineered? *4GL* - Does the tool incorporate a fourth-generation language of its own to enable links with the 'outside' world to be created?
DBMS - Can a physical design be generated for various target DBMS (relational, hierarchical and/or network)?
Architecture - Is the architecture of the system open so that the working environment can be customized and extended? Does the format of the repository enable integration across various CASE products?
Hardware requirements - What is the hardware configuration required running the software?
Multi-user access - Can the members of project teams share common data? Can data be shared between project teams? What level of record locking exists in the repository? What security controls exist governing single-user and multi-user access? How is individual work interfaced into the system?
Performance - If there is code generation, does the code perform efficiently?
Suitability - What academic level is the tool suitable for? How easily will it fit into the current work procedures? How much training will be required? For how long will staff be unproductive while learning to use the tool? How long will it take the students to learn how to use the tool?
Cost - How much is it? Does the vendor have an educational license agreement? What are the maintenance arrangements?
Vendor - Is the CASE product distributed in Australia? How long has the vendor been selling the CASE tool? Does the vendor have staff who is experienced with the software? Does the vendor provide training, maintenance and upgrades and at what cost?
Future - What are the prospects for the future development of the tool? Some issues here are distributed database support, the use of expert systems, cooperative processing support, graphical user interface, portability, heterogeneous systems gateways, and security.
Performance - If there is code generation, does the code perform efficiently?

3.6.1 CASE SOFTWARE DEVELOPMENT ENVIRONMENT

CASE tools support extensive activities during the software development process. Some of the functional features that are provided by CASE tools for the software development process are(Ref. 09):

1. Creating software requirements specifications
2. Creation of design specifications
3. Creation of cross references.
4. Verifying/Analysing the relationship between requirement and design
5. Performing project and configuration management
6. Building system prototypes
7. Containing code and accompanying documents.
8. Validation and verification, interfacing with external environment.

Some of the major features that should be supported by CASE development environment are:
- a strong visual support
- prediction and reporting of errors
- generation of content repository
- support for structured methodology
- integration of various life cycle stages
- consistent information transfer across SDLC stages
- automating coding/prototype generation.

3.6.2 CASE TOOLS AND REQUIREMENT ENGINEERING

Some of the queries are to be made like;
Which is the most Common Source of Risk during the process of software development?

One of the major risk factors that affect project schedule, budget and quality can be defined as the ability to successfully elicit requirements to get a solution. Statistically it has been seen that about 80% of rework in projects is due to requirement defects. How can CASE tools help in effective Requirements Engineering (RE)?Good and effective requirements engineering tool needs to incorporate the best practices of requirements definition and management. The requirements engineering approach should be highly iterative with the goal of establishing managed and effective communication and collaboration(Ref. 04).

Thus, a CASE tool must have the following features from the requirements engineering viewpoint:
- a dynamic, rich editing environment for team members to capture and manage requirements
- to create a centralized repository
- to create task-driven workflow to do change management, and defect tracking.

But, what is a good process of Requirements Engineering for the CASE?
A simple four-step process for requirements engineering
Requirement Elicitation
A simple technique for requirements elicitation is to ask “WHY”.
CASE tools support a dynamic, yet intuitive, requirements capture and management environment that supports content and its editing. Some of the features available for requirement elicitation are:
- Reusable requirements and design templates for various types of system
- Keeping track of important system attributes like performance and security
- It may also support a common vocabulary of user-defined terms that can be automatically highlighted to be part of glossary.
- Provide feature for the assessment of the quality of requirements

• Separate glossary for ambiguous terms that can be flagged for additional clarification.

What do we expect from the tool?

• It should have rich support for documentation that is easily understandable to stakeholders and development teams.

• It should have the capability of tracking of requirements during various SDLC systems.

• It should help in the development of standard technical and business terminology for end clients.

Software Analysis and Specification :

One of the major reasons of documenting requirements is to remove the ambiguity of information. A good requirement specification is testable for each requirement.

One of the major features supported by CASE tools for specification is that the design and implementation should be traceable to requirements. A good way to do so is to support a label or a tag to the requirements. In addition it should have the following features:

• Must have features for storing and documenting of requirements.

• Enable creation of models that are critical to the development of functional requirements.

• Allow development of test cases that enable the verification of requirements and their associated dependencies.

• Test cases help in troubleshooting the correlation between business requirements and existing system constraints.

CASE and Web Engineering

CASE Tools are also very useful in the design, development and implementation of web site development.

Web Engineering requires tools in many categories. They are:

• Site content management tools
• Site version control tools
• Server management tool
• Site optimization tools
• Web authoring and deployment tools
• Site testing tools that include load and performance testing
• Link checkers
• Program checkers
• Web security test tools.

Validation of Requirements

A very important feature in this regard is to allow collaboration yet customizable workflows for the software development team members. Also facilitating approvals and electronic signatures to facilitate audit trails. Assigning owner of requirement may be helpful if any quality attributes may need changes. Thus, a prioritized validated documented and approved requirements can be obtained.

Managing Requirements

The requirements document should have visibility and help in controlling the software delivery process. Some such features available in CASE tools are:

• estimation of efforts and cost

• specification of project schedule such as deadline, staff requirements and other constraints

• specification of quality parameters.

Software Change Management

An incomplete or ambiguous requirement if detected early in the analysis phase can be changed easily with minimum cost. However, once they are converted to baselines after requirements validations the change should be controlled.

One of the major requirements of change management is:

Any change to these requirements should follow a process which is defined as the ability to track software requirements and their associated models/documents, etc. (e.g., designs, DFDs and ERDs, etc.). This helps in determining the components that will be impacted due to change. This also helps tracking and managing change. The whole process starts with labeling the requirements properly.

Most CASE Tools store requirement baselines, including type, status, priority and change history of a software item. Such traceability may be bi-directional in nature.

Static word processing documents or spreadsheet recedes communication issues because those tools do not allow sharing and collaboration on up-to-date requirements. To overcome this a CASE enabled requirements management infrastructure offers the familiarity of environments such as Microsoft Word, with added communication methods, such as email. Thus, CASE is a very useful tool for requirement engineering.

3.6.3 CASE TOOLS AND DESIGN AND IMPLEMENTATION

In general, some CASE tools support the analysis and design phases of software development. Some of the tools supported by the design tools are(Ref. 14):

• Structured Chart.

• Program Document Language (PDL).

• Optimization of ER and other models.

• Flow charts.

• Database design tools.

• File design tools.

Some of functions that these diagrams tool support are simple but are very communicative as far as representations of the information of the analysis and design phases are concerned. CASE Tools also support standard representation of program architecture; they also contain testing items related to the design and debugging. Automatic support for maintenance is provided in case any of the requirements or design items is modified using these diagrams. These CASE tools also support error-checking stages. They allow checks for completeness and consistency of the required functional design types and consistency at various levels of cross referencing among the documents consistency checking for these documents during the requirements analysis and design phases.

Proper modeling helps in proper design architecture. All the CASE tools have strong support for models. They also help in resolving conflicts and ambiguity among the models and help in optimizing them to create excellent design architecture and process implementation.

CASE REPOSITORY

CASE Repository stores software system information. It includes analysis and design specifications and helps in analyzing these requirements and converting them into program code and documentation. The following is the content of CASE repository:

Data
Process
Models
Rules/ Constraints.

Data: Information and entities/object relationship attributes, etc.
Process: Support structured methodology, link to external entities, document structured software development process standards.
Models: Flow models, state models, data models, UML document etc.
Rules/Constraints: Business rules, consistency constraints, legal constraints.
The CASE repository has two primary segments.

1. Information repository
2. Data dictionary.

Information Repository includes information about an organization's business information and its applications.
The CASE tools manage and control access to repository. Such information data can also be stored in corporate database.
Data dictionary contains all the data definitions for all organizational applications, along with cross-referencing if any.
Its entries have a standard definition, viz., element name and alias; textual description of the element; related elements; element type and format; range of acceptable values and other information unique to the proper processing of the element.
CASE Repository has additional advantages such that it assists the project management tasks; aids in software reusability by enabling software modules in a manner so that they can be used again.

IMPLEMENTATION TOOLS AND CASE

CASE tools provide the following features for implementation:

• Diagramming tools enable visual representation of a system and its components
• Allow representing process flows.
• Help in implementing the data structures and program structures
• Support automatic creation of system forms and reports.
• Ready prototype generation.
• Create of both technical and user documentation.
• Create master templates used to verify documentation and its conformance to all stages of the Software Development Life Cycle (SDLC).
• Enable the automatic generation of program and database from the design documents, diagrams, forms and reports stored in the repository.

3.6.4 SOFTWARE TESTING

CASE tools may also support software testing. One of the basic issues of testing is management, execution and reporting. Testing tools can help in automated unit testing, functional regression testing, and performance testing. A testing tool ensures the high visibility of test information, types of common defects, and application readiness(Ref. 10).

Features Needed for Testing
The tool must support all testing phases, viz., plan, manage and execute all types of tests viz., functional, performance, integration, regression, testing from the same interface.
It should integrate with other third party testing tools.
It should support local and remote test execution.
It should help and establish and manage traceability by linking requirements to test cases. This also helps when requirements change; in such as case the test cases traced to the requirement are automatically flagged as possible candidates for change.

3.6.5 SOFTWARE QUALITY AND CASE TOOLS

Software quality is sacrificed by many developers for more functionality, faster development and lower cost. However, one must realize that a good quality product actually enhances the speed of software development. It reduces the cost and allows enhancement and functionality with ease as it is a better structured product. You need to pay for a poor quality in terms of more maintenance time and cost. Can the good quality be attributed to software by enhancing the testing? The answer is NO. The high quality software development process is most important for the development of quality software product. Software quality involves functionality for software usability, reliability, performance, scalability, support and security.
Integrated CASE tools:

• help the development of quality product as they support standard methodology and process of software development

• supports an exhaustive change management process

• contains easy to use visual modeling tools incorporating continuous quality assurance.

Quality in such tools is represented in all life cycle phases viz., Analysis/ Design development, test and deployment. Quality is essential in all the life cycle phases.
Analysis: A poor understanding of analysis requirements may lead to a poor product. CASE tools help in reflecting the system requirements clearly, accurately and in a simple way. CASE tools support the requirements analysis and coverage also as we have modeling. CASE also helps in ambiguity resolution of the requirements, thus making high quality requirements.
Design: In design the prime focus of the quality starts with the testing of the architecture of the software. CASE tools help in detecting, isolating and resolving structure deficiency during the design process. On an average, a developer makes 100 to 150 errors for every thousand lines of code. Assuming only 5% of these errors are
serious coding errors in your system. One of the newer software development processes called the Agile process helps in reducing such problems by asking the developer to design their test items first before the coding.
A very good approach that is supported by CASE tools specially running time development of C, C++, JAVA or .NET code is to provide a set of automatic run time
Language tools for development of reliable and high performance applications.
Testing: Functionality and performance testing is an integrated part of ensuring high quality product. CASE support automated testing tools that help in testing the software, thus, helping in improving the quality of testing. CASE tools enhance the speed breadth and reliability of these design procedures. The design tools are very important specifically in case of a web based system where scalability and reliability are two major issues of design.

Deployment: After proper testing a software goes through the phase of deployment where a system is made operational. A system failure should not result in a complete failure of the software on restart. CASE tools also help in this particular place. In addition, they support configuration management to help any kind of change thus to be made in the software.
Quality is teamwork: It involves integration of workflow of various individuals. It establishes a traceability and communication of information, all that can be achieved by sharing workload documents keeping their configuration items.

3.6.6 SOFTWARE CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) is extremely important from the view of deployment of software applications. SCM controls deployment of new software versions. Software configuration management can be integrated with an automated solution that manages distributed deployment. This helps companies to bring out new releases much more efficiently and effectively. It also reduces cost, risk and accelerates time(Ref. 03).

A current IT department of an organization has complex applications to manage. These applications may be deployed on many locations and are critical systems. Thus, these systems must be maintained with very high efficiency and low cost and time. The problem for IT organizations has increased tremendously as some of the organizations may need to rebuild and redeploy new software versions a week over multi-platform global networks.
In such a situation, if rebuilding of application versions is built or deployed manually using a spreadsheet, then it requires copying of rebuilt software to many software locations, which will be very time consuming and error prone. What about an approach where newer software versions are automatically built and deployed into several distributed locations through a centralized control? Thus, an automatic deployment tool will be of great use under the control of SCM.

3.6.7 SOFTWARE PROJECT MANAGEMENT AND CASE TOOLS

Developing commercial software is not a single-player activity. It is invariably a collaborative team activity. The development of business-critical systems is also too risky, or is too large for a single developer to deliver the product in a stipulated time and quality frame. A software team involves designers, developers, and testers who work together for delivering the best solution in the shortest time. Sometimes, these teams can be geographically dispersed. Managing such a team may be a major change requests, and task management(Ref. 13).
The CASE tools help in effective management of teams and projects. Let us look into some of the features of CASE in this respect:

- Sharing and securing the project using the user name and passwords.
- Allowing reading of project related documents
- Allowing exclusive editing of documents
- Linking the documents for sharing
- Automatically communicative change requests to approver and all the persons who are sharing the document.
- You can read change requests for yourself and act on them accordingly.
- Setting of revision labels so that versioning can be done.
- Any addition or deletion of files from the repository is indicated.
- Any updating of files in repository is automatically made available to users.

• Conflicts among versions are reported and avoided
• Differences among versions can be visualized.
The linked folder, topics, and change requests to an item can be created and these items if needed can be accessed.
• It should have reporting capabilities of information
The project management tools provide the following benefits:
• They allow control of projects through tasks so control complexity.
• They allow tracking of project events and milestones.
• The progress can be monitored using Gantt chart.
• Web based project related information can be provided.
• Automatic notifications and mails can be generated.

Some of the features that you should look into project management software are:
It should support drawing of schedules using PERT and Gantt chart.
It should be easy to use such that tasks can be entered easily, the links among the tasks should be easily desirable.
Milestones and critical path should be highlighted.
It should support editing capabilities for adding/deleting/moving tasks.
Should map timeline information against a calendar.
Should allow marking of durations for each task graphically.
It should provide views tasks, resources, or resource usage by task.
Should be useable on network and should be able to share information through network.

IV. SOLVING THE CHALENGES AND IMPROVING THE DSD STRENGTH

From the experimental studies analyzed, we have extracted the following success factors of DSD. The primary studies referenced are listed in the appendix.
(i)Intervention of human resources by participating in surveys.
(ii)Carrying out improvements based on the needs of the company, taking into account the technologies and methodologies used. The tools employed at the present must be adapted and integrated.
(iii)Training of human resources in the tools and processes introduced.
(iv) Registration of activities with information on pending issues, errors and people in charge, and the provision of awareness of software development activities.
(v)Establishment of an efficient communication mechanism between the members of the organization, allowing a developer to discover the status and changes made within each project.
(vi) Using a version control tool in order to control conflictive situations.
(vii)There must be a manner in which to permit the planning and scheduling of distributed tasks, taking into account costs and dependencies between projects, and the application of corrective measures and notifications.
(viii) Application of maturity models and agile methodologies based on incremental integration and frequent deliveries.
(ix)Application of MDD approaches to automate development tasks.
(x)Systematic use of metrics tailored to the organization.
In this work we have applied a systematic review method in order to analyze the literature related to the topic of DSD within the FABRUM project context whose main objective is to

create a new DSD model with which to manage the relationships between a planning and design center and a software production factory. This work serves as a starting point from which to establish the issues upon which subsequent research will be focused. The results obtained from this systematic review have allowed us to obtain a global vision of a relatively new topic which should be investigated in detail. However, every organization has concrete needs which basically depend on its distribution characteristics, its activity and the tools it employs. These factors therefore cause this subject to be extremely wide-ranging, and lead to the necessity of adapting both the technical and organizational procedures, according to each organization's specific needs. The proposals found in the analyzed studies were, in general, mainly concerned with improvements related to the use of collaborative tools, the integration of existing tools, source code control, or the use of collaborative agents. Moreover, it should be stressed that the evaluation of the results obtained from the proposed improvements are often based on studies in a single organization, and sometimes only takes into account the developers' subjective perception. On the other hand, it should be noted that maturity models such as CMM, CMMI, or ISO, which would be of particular relevance to the present investigation, represent only 17% of all analyzed works. The fact that almost all the experimental studies that employed CMMI and CMM applied a maturity level of 2 suggests that the cost of implementing higher maturity levels in distributed environments might be too high. However, there is a need for more studies related to the application of maturity models and metrics to quantify issues related to the process areas. The application of agile methodologies based on incremental integration and frequent deliveries, and frequent reviews of problems to adjust the process become important success factors. We also found an increasing interest in modeling in software development, and MDA approaches as a means to improve productivity, quality and understanding among members involved in the development process(Ref. 07). Finally, we must emphasize that the search was reduced to a limited number of search engines and excluded studies which addressed the subject of DSD but did not contribute any significant method or improvement in this research context. However, since this is such a wide area, some of these works present interesting parallel subjects for the development of this investigation, and their study would, therefore, be important in a future work. We also have found studies related to the business perspective or focused on the customer which may be useful for related works. Furthermore, many studies mainly related to tools which are not included in the context of DSD but are useful in fields related to communications or source control also exist.

REFERENCES

[1] Cheetam, G. and Chivers, G. (2005). Professions, Competence and Informal Learning. Edgard Elgar Publishing Limited.

[2] Colucci, S., Noia, T. D., Sciascio, E. D., Donini, F. M.,Mongiello, M., and Mottola, M. (2003). A formal approach to ontology-based semantic match of skills descriptions.J. UCS, 9(12):1437–1454.

[3] de Bruijn, J. and Polleres, A. (2004). Towards an ontology mapping specification language for the semantic web.

[4] DERI Technical Report.HR-XML (2004). HR-XML Measurable Competencies. http://www.hr-xml.org.

[5] IEEE RCD (2005). IEEE 1484.20.1/draft - draft standard for Reusable Competency Definitions (RCD).

[6] http://ieeeltsc.org/wg20Comp/Public/IEEE_1484.20.1.D3.pdf.

[7] IMS RDCEO (2002). IMS Reusable Definition of Competency or Educational Objective (RDCEO). http://www.imsglobal.org/competencies/.

[8] Lau, T. and Sure, Y. (2002). Introducing ontology-based skills management at a large insurance company. In Modellierung 2002, Modellierung in der Praxis – Modellierung f¨ur die Praxis, pages 123–134, Tutzing,Deutschland.

[9] Lefebvre, B., Gauthier, G., Tadi´e, S., Duc, T. H., and Achaba,H. (2005). Competence ontology for domain knowledge dissemination and retrieval. Applied Artificial Intelligence,19(9-10):845–859.

[10] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350.

[11] Riehle, D. (1997). Composite design patterns. In ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications, pages 218–228.

[12] SRCM (2006). Simple Reusable Competency Map proposal (SRCM). http://www.ostyn.com/resources.htm.

[13] Sure, Y., Maedche, A., and Staab, S. (2000). Leveraging corporate skill knowledge - from proper to ontoproper. In Mahling, D. and Reimer, ., editors, 3rd International Conference on Practical Aspects of Knowledge Management,Basel, Switzerland.

[14] Webster. Webster online dictionary. http://www.webster.com/.Wikipedia. Wikipedia encyclopedia. http://en.wikipedia.org/.

[15] S.Saira Thabasum, "Need for Design Patterns and Frameworks for Quality Software Development", International Journal of Computer Engineering & Technology (IJCET), Volume 3, Issue 1, 2012, pp. 54 - 58, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.

[16] Shobha B. Patil and S.K. Shirgave, "Enriching Search Results using Ontology", International Journal of Computer Engineering & Technology (IJCET), Volume 4, Issue 2, 2013, pp. 500 - 507, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.

[17] Dillip Kumar Mahapatra, Tanmaya Kumar Das and Gopakrishna Pradhan, "Guidelines for Managing Distributed Software Project Under Deployment", International Journal of Computer Engineering & Technology (IJCET), Volume 4, Issue 1, 2013, pp. 34 - 45, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.