IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727Volume 8, Issue 3 (Jan. - Feb. 2013), PP 13-28
www.iosrjournals.org

Scope of Case Tools for Managing Distributed Software Development: A Proposal for Using Distributed Software Product in Different Organizational Conditions

Dillip Kumar Mahapatra[1], Tanmaya Kumar Das[2], Gopa Krishna Pradhan[3]
[1](Department of Information Technology, Krupajal Engineering College, Bhubaneswar, Odisha, India / Biju Pattanaik University of Technology, Rourkela, Odiaha, India.
[2] (Department of Computer Science & Engineering, C.V. Raman College of Engineering, Bhubaneswar, Odisha, India / Biju Pattanaik University of Technology, Rourkela, Odiaha, India.
[3] (Ex-Prof.Department of Computer Science & Engineering, S O A University, Bhubaneswar, Odisha, India

Abstract: *Software development is an intense collaborative process where success depends on the ability to create, share and integrate information. Given the trend towards globalization in the software development industry, the complexity of distributed systems has increased, which needs a systematic approach to the development of distributed systems very important. This paper introduces taxonomy of software engineering tools for distributed projects and presents collaborative development environments, ranging from classic platforms for dispersed developers in open source software projects to modern environments for flexible and distributed processes. It eases the distributed system development, increases the portability of the software and improves the system maintenance and reliability.*

Keywords: *Distributed software development, Software Process, Computer-Aided Software Engineering Tools, collaborative development environments, Distributed Software Project Management, Tools integration.*

I. Introduction

Distributed software development takes a concurrent rather than sequential development approach, effectively using iteration to show progress and manage risk. This provides a basis for more rapid development, with smooth transitioning from one stage of a project to the next as well as continuous delivery of staged results that are of value in the total solution. The use of this approach implies iteration with a checkpoint at the conclusion of each stage to validate the quality of stage results, as well as the scope of the next stage. In addition to concurrency across stages, a release-based strategy ensures that there are short intervals between incremental releases of tangible results. This ensures that the direction of the project can be adjusted dynamically to accommodate critical events and needs.

Working across distances has become common place for software projects today. Nevertheless, distance creates an additional challenge to development processes, because of fewer opportunities for rich interaction and lower frequencies of direct communication. In order to support collaborative work on their projects, software engineers communicate both directly, through meetings and informal conversations, and indirectly, by means of software artifacts. Adequate tool support is paramount to enable collaboration in distributed software development. However, most work in collaborative environments for distributed development has focused on code-specific tasks rather than on other software engineering activities at a higher level of abstraction like requirements engineering or software design. Collaboration in Distributed Software Development scenario is significantly affected by the stakeholders, developers teams geographical distributed. There is a need to further our knowledge of what is the most appropriate collaboration tool set to achieve a shared understanding among distributed project stakeholders, then distributed software development can be benefited from an interdisciplinary approach.

This chapter is organized to provide the motivation for using different CASE tools through the entire distributed software development process, ranging from user requirements, feasibility study .to maintenance and re-engineering (Ref. 01).

II. Distributed Software Development Process

The Distributed Software Development Process is the result of a commitment to continuous learning, refinement of experience and improvement of process as new ideas have been developed and new technologies have been employed. It is a collection of experiences and best practices that have been taken from real-world development engagements, providing development teams with access to shared experiences and a proven, repeatable process. This DSD Process encompasses modern design principles and proven practices to facilitate

the development task and provide developers with a blueprint for building robust and correct distributed applications.

Technically, distributed development is based on a multi-tiered development architecture. In its simplest form, with two tiers, a distributed application is synonymous with a client/server protocol in which you use a set of rules that specify a behavior for two collaborating processes. In a client/server relationship, one process (the client) initiates the interaction by issuing a request to a second process (the server). The server process must await a request from the client and, on receipt of that request, performs a service and returns a response (or result) to the client. The server is capable of handling requests from multiple clients and is responsible for coordinating and synchronizing responses.

Although the technical definition of the client/server protocol is stated in terms of a relationship between software processes, a more popular definition describes the technical architecture that supports the software. Client/server architecture provides an opportunity to distribute an application across two or more computers that can be used most effectively to deliver departmental and enterprise-wide systems solutions.

In a three-tiered architecture (a distributed architecture), an additional layer (the middle layer) is used to accept, process and mediate requests from the client to the server. The middle layer serves to alleviate the processing of rules and decision logic from both the client and the server. This permits the construction of "thin clients" and the removal of processing logic from the server layer. The server layer then can behave as a source of raw information. Processes can be distributed to any one of the layers(Ref. 03).

Distributed architecture is based on a network model in which processes can be distributed on any processor and any two individual nodes of the network are in a client/server relationship with any number of intervening middle layers. The heart of distributed architecture is based on the client/server pattern. The additional complexity comes from the design of the components for the middle layer, referred to as "middleware." Distributed software technology has much in common with and is often served by object technology and software components. There are many points of synergy between these technologies, including a focus on real world modeling, as well as a focus on simplicity, reusability, extensibility, and productivity.

Distributed development is based on several key elements:

- Concurrent development of packages and components
- Reuse of software components (either built in-house or purchased)
- Cyclical and incremental development
- Release strategy

Distributed Software development takes a concurrent rather than sequential development approach, effectively using iteration to show progress and manage risk. This provides a basis for more rapid development, with smooth transitioning from one stage of a project to the next as well as continuous delivery of staged results that are of value in the total solution. The use of this approach implies iteration with a checkpoint at the conclusion of each stage to validate the quality of stage results, as well as the scope of the next stage. In addition to concurrency across stages, a release-based strategy ensures that there are short intervals between incremental releases of tangible results. This ensures that the direction of the project can be adjusted dynamically to accommodate critical events and needs.

Distributed Software Development Process Relies Upon Three Distinct Architectures:

Peer –to-Peer / One tier: All the processing is done on only one machine,& number of clients attached to this machine.(Mainframe)

***Two tier*:** The client process runs on a workstation or personal computer that interacts with a server process which runs on a shared device that is accessed through a network.

***Three tier*:** The client process runs on a client workstation that interacts with a server process which runs on a server device. The server device is connected to a host that provides services to the server device.

N-tier: The client process runs on any workstation; the server process runs on one or more distributed server devices. The middleware mediates all interactions between the various processes. Components and integration adapters allow access to various information sources.

Distributed implementations are based on multiple levels of complexity, all of which are characterized by the distribution of processing logic. The choice of hardware architecture, development tools, and the development approach is influenced by the distribution strategy that results from the business requirements and system characteristics. Distributed development strategies are defined in terms of distributing presentation, function, and data access logic across the client and the server. The presentation layer handles the user interface and formatting information for display. The business object layer executes the business logic and is based on core business objects. The data access layer handles data-related processing. Distributed solutions rely, in part, on other technologies such as a graphical user interface (GUI) or Web-based User Interface. The user interface is the external view of the system and, by definition, is the basis for most users' judgment of the system. The

presentation system, the development tools, and the skill of the designer define the functionality of the user interface.

Networking is a principal component of distributed solutions. Whether served by a local area network (LAN), wide area network (WAN), or Web-based global network, distributed architectures generally include a network that links workstations to servers and servers to hosts.

One or more databases will be present in almost all distributed systems. Increasingly, the use of intelligent databases that support some form of processing in the database itself is the norm. This intelligence takes the form of triggers or stored procedures that provide processing logic that is independent of the programs that access it. Data warehouses, data marts and intelligent databases augment those basic mechanisms.

The use of distributed database technology is also a common part of the design. Distributed database technology permits a single logical database to reside on different computers, linked by a network. This enables the database to behave as a single local entity, even though it is distributed, providing for economies in network traffic and improved performance.

Factors that measure the success of the implementation of DSD technology are many and varied. They include:

***Usability*.** This includes improving user productivity and satisfaction with the system through easier access to and manipulation of data, consistency in the user interface, and access to data through end user tools.

***Reusability*.** This measures the capacity to reuse components developed for the application in other distributed applications with a common interface.

Stability. Distributed technology is becoming more robust but is still relatively immature. Consequently, it is essential that the known weaknesses in this technology be acknowledged, understood, sought out and corrected through the design process to achieve stable applications.

Consistency. The single greatest factor contributing to user satisfaction is consistency. Users expect that the same types of actions will occur in the same way each time they encounter them. The challenge is to provide a consistent work environment that ensures consistency across systems and tools.

Quality. Quality takes on special meaning in distributed implementations, due to the complexities in distributing and upgrading applications across a network environment. Quality must be built in by applying a robust methodology and testing for quality at each checkpoint in the process.

Distributed technology is powerful; however, it is not a panacea for all situations. There are application characteristics that suggest a distributed application solution and some that suggest that the risk is too high. Some applications play to the strengths of distributed technology - some do not.

***Strengths* of distributed technology include:**

- Graphical presentation to the users
- Network distribution of applications
- Integration of legacy or heritage applications
- Integration of purchased packages and components
- Robust application based on real-world business objects
- Frameworks and patterns availability to reduce development time
- Access to shared business data

***Weaknesses* include:**

- Complex or long update transaction cycles
- Distributed transaction update
- Very large database support on servers

The following diagram depicts the dependency relationships between the stages in the distributed software development process (Ref. 02).

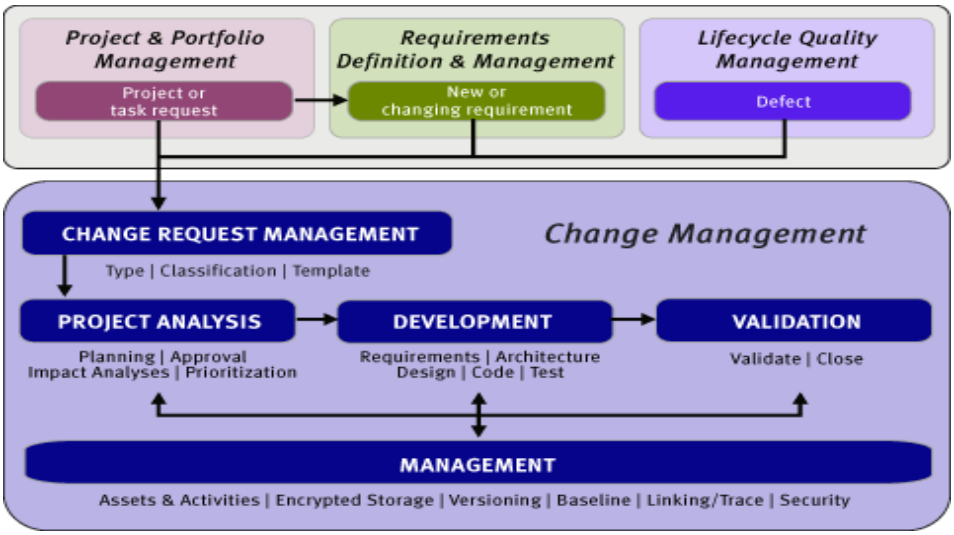


Fig.1: Stages of DSD process

III. Scope Of Computer Assisted Software Engineering (Case) Tools

Computer Assisted Software Engineering (CASE) tools were developed to support the professional system developer and improve their productivity in the complex task of developing large information systems.From a developer's viewpoint, CASE tools provide support for modelling aspects of the system using a variety of notations and techniques: from diagrams to mathematics and text, producing prototype code, and even verifying the correctness of the system design. It is also sometimes tedious process of writing system documentation and keeping it up to date. Tools often allow the creation and management of a central repository of documents and other artefacts, which are useful both for communication within a development team, and for project management and decision tracking. Importantly, they improve the quality of the development processes by supporting, and to a large extent enforcing, a standard methodology and sound design principles.

Despite of these potential benefits, there are often constraints to the adoption of tools. The introduction of a CASE tool within an organisation comes at a high cost: an upfront investment is required to acquire the technology and to train personnel accordingly, while the benefits of using the tool manifest themselves only in the long term. CASE tools are only effective if standard methodologies and processes are adopted across the organisation; the definition of standard procedures and practices also has to be established at the beginning. At times the obstacles are cultural: there may be some resistance from analysts and developers who perceive the rigidity of an enforced methodology as a threat to their autonomy and creativity. The initial learning curve is quite steep and they may not consider the effort worthwhile in terms of improvement in productivity and quality. Another major drawback is that the available tools tend to be narrow in their focus and concentrate on small subset of activities within the development process. For instance, most tools support modelling, while hardly any provide facilities for communication between customers and developers. This may result in a variety of tools to be used within a process, with all the difficulties arising from integration and interchange of information (Ref. 04).

Usually a question strike in our mind that ***which factors may contribute to the successful adoption of CASE tools within an organisation****?*

The answer to the above question is as following:

- There should be established methods and procedures across the organisation that can be supported by the tools.
- Adequate investment should be put into training managers and developers.
- Deployment should occur over a long period of time, as the benefits of CASE tools are not short term.
- Clear procedures should be established for the use of different tools within development and standards should be followed for information exchange among tools.
- To overcome some of the deficiencies of current tools, they should be integrated within practices that support project management and customer/developer communication.

A great variety of tools are available today, supporting many different functions within software development.

Table-1 shows a brief taxonomy of tools based on their function; the list is not meant to be exhaustive. Also, many current CASE systems integrate many such functions within the same application environment.

Table 1: Taxonomy of CASE tools

TOOLS	FUNCTIONS
Process management tools	To capture and model development processes
Project management tools	To plan and schedule projects, and track progress
Risk analysis tools	To record, categorize and analyze risks
Requirements management tools	To capture, analyze and trace requirements throughout development
Metrics tools	To capture specific software metrics and provide overall measures of quality
Modeling tools	To support modeling within analysis and design activities
Programming tools	To support code development
Interface design tools	To support the design of graphical use interfaces (GUI)
Test management tools	To manage and coordinate testing
Process management tools	To capture and model development processes

1.1. Process management Tools

Process management is a series of techniques, skills, tools, and methods used to control and manage a business process within a large system or organization. The term is most commonly used in business analysis, and systems engineering. The purpose of process management is to clearly identify and document all the steps and actions taken to complete a process or work flow. This type of work requires a great attention to detail, excellent written communication skills, analysis skills and the ability to objectively meet the requirements of the project. Process management involves documenting the current process, evaluating of time and level of effort, as well as analysis of efficiency, bottlenecks, and overall process costs. Exercises in re-engineering often start with process management and analysis.

Business processes are dependant and ordered activities that result in predictable and repeatable outcomes. Consisting of an organization's operating procedures, institutional working knowledge, and information resources, business processes are designed to satisfy defined business objectives in an efficient and timely manner. In an efficient environment, the functional components of a process can be readily identified, adapted, and deployed to address ever-changing corporate requirements—a capability termed as business agility. By definition, business agility is an organization's systemic ability to fluidly marshal and reconfigure resources in response to business requirements and opportunities. Business Process Management (BPM) tools are designed to provide for such agility by facilitating the creation and execution of highly transparent and modular process-oriented workflows that meet the operational performance standards IT organizations demand.

Automated business processes developed and executed within such an environment are characterized by the following attributes:

- Visibility of end-to-end process activities
- Process components and functionality that are exposed and self-describing
- Ability to integrate disparate information source and application functionality into a process
- Information flow and event notification that can be automated and monitored throughout a process
- Workflow participation that makes the most of desktop productivity and communication tools
- Service level agreements that can be specified, monitored, and enforced for activities in a process
- Ability to add, remove, or reconfigure any process activity or component, without disrupting the process
- Processes that can be monitored in real time or near real time
- Process designs that can accommodate any exception handling requirement
- Processes that can be easily replicated, extended, and scaled

With the support of XML and Web Services, BPM systems are transforming the way in which IT organizations are implementing and executing workflow components. XML applies structure to information, freeing it from any functional dependency on the software that operates on it. Web Services on the other hand provide the framework for application-to-application messaging and invocation over an unbounded network. BPM tools provide the additional support infrastructure to harness these capabilities to create, deploy, and execute the entire scope of workflow management, enterprise application integration (EAI), and trading partner integration (TPI).

Some of the Process Management tools are;

Tools	Features					
	Process domain	Process life-cycle support	Process modeling language	Other supported technologies	Development support	Usefulness for engineers
Eclipse Process Composer (EPFC)	Software	Modeling	UMA (SPEM-based)	n/a	Knowledge, sharing, communication	Publish processes as website; knowledge base
OpenUP	Software	Modeling (restricted)	UMA (SPEM-based)	n/a	Guidance for RUP	n/a
Rational Method Composer (RMC)	Software	Modeling	UMA (SPEM-based)	n/a	Knowledge, sharing, communication	Publish processes as website
Appian Enterprise BPM Suite	Business	Modeling, collaboration, automation	BPMN	n/a	Knowledge, sharing, communication	Enterprise integration of process and content management; collaborative team workspaces and tools
ARIS	Business	Modeling, collaboration, automation	BPMN	EPC, UML, BPEL, and WSDL	Knowledge, sharing	Web-based design; decentralized Web-based management of IT resources
BizAgi	Business	Modeling, collaboration, automation	BPMN	BPEL, SOA, XPath, Tempo, WS-Human Task Service, J2EE, .NET	Knowledge, sharing, communication	Process publication in website; concurrent process working; interaction with other systems via SOA and SharePoint WebParts
Intalio	Business	Modeling, collaboration, automation	BPMN	BPEL, J2EE, SOA	Knowledge, sharing, communication	Process publication; collaboration facilities for process management; Intalio Social Portal cloud computing platform

Table 2: Process Management tools

Process management tools support software teams in three ways:

• Modelling.

Define consistent process models, thereby reducing any ambiguity about how the work should be done be- cause well-defined and documented pro- cesses must be defined and consistently followed across all sites.

• Collaboration

Share information and knowledge among different sites, thereby avoiding repeated mistakes or inconsistencies between different versions of information.

• Automation

Provide tools to automate a process by means of workflows, thereby increasing engineering and development productivity across the life cycle because you're sharing information across tools, orchestrating interfaces, and maintaining work products.

1.2. Project management tools

The project management process typically includes four key phases: initiating the project, planning the project, executing the project, and closing the project. An outline of each phase is provided below.

Initiating the Project: The project management techniques related to the project initiation phase includes:

1. ***Establishing the project initiation team***. This involves organizing team members to assist in carrying out the project initiation activities.
2. ***Establishing a relationship with the customer.*** The understanding of your customer's organization will foster a stronger relationship between the two of you.
3. ***Establishing the project initiation plan.*** Defines the activities required to organize the team while working to define the goals and scope of the project.
4. ***Establishing management procedures.*** Concerned with developing team communication and reporting procedures, job assignments and roles, project change procedure, and how project funding and billing will be handled.
5. ***Establishing the project management environment and workbook.*** Focuses on the collection and organization of the tools that you will use while managing the project.

***Planning the Project*:** The project management techniques related to the project planning phase include:

1. *Describing project scope, alternatives, and feasibility.* The understanding of the content and complexity of the project. Some relevant questions that should be answered include:
   - What problem/opportunity does the project address?
   - What results are to be achieved?
   - What needs to be done?
   - How will success be measured?
   - How will we know when we are finished?

2. ***Divide the project into tasks***. This technique is also known as the work breakdown structure. This step is done to ensure an easy progression between tasks.
3. ***Estimating resources and creating a resource plan.*** This helps to gather and arrange resources in the most effective manner.
4. ***Developing a preliminary schedule.*** In this step, you are to assign time estimates to each activity in the work breakdown structure. From here, you will be able to create the target start and end dates for the project.
5. ***Developing a communication plan***. The idea here is to outline the communication procedures between management, team members, and the customer.
6. ***Determining project standards and procedures***. The specification of how various deliverables are produced and tested by the project team.
7. ***Identifying and assessing risk.*** The goal here is to identify potential sources of risk and the consequences of those risks.
8. ***Creating a preliminary budget.*** The budget should summarize the planned expenses and revenues related to the project.
9. ***Developing a statement of work.*** This document will list the work to be done and the expected outcome of the project.
10. ***Setting a baseline project plan.*** This should provide an estimate of the project's tasks and resource requirements.

***Executing the Project*:** The project management techniques related to the project execution phase include:

1. ***Executing the baseline project plan***. The job of the project manager is to initiate the execution of project activities, acquire and assign resources, orient and train new team members, keep the project on schedule, and assure the quality of project deliverables.
2. ***Monitoring project progress against the baseline project plan***. Using Gantt and PERT charts, which will be discussed in detail further on in this paper, can assist the project manager in doing this.
3. ***Managing changes to the baseline project plan.***
4. ***Maintaining the project workbook.*** Maintaining complete records of all project events is necessary. The project workbook is the primary source of information for producing all project reports.
5. ***Communicating the project status.*** This means that the entire project plan should be shard with the entire project team and any revisions to the plan should be communicated to all interested parties so that everyone understands how the plan is evolving.

Closing down the Project: The project management techniques related to the project closedown phase include:

1. ***Closing down the project.*** In this stage, it is important to notify all interested parties of the completion of the project. Also, all project documentation and records should be finalized so that the final review of the project can be conducted.
2. ***Conducting post project reviews***. This is done to determine the strengths and weaknesses of project deliverables, the processes used to create them, and the project management process.
3. ***Closing the customer contract.*** The final activity is to ensure that all contractual terms of the project have been met.

The techniques listed above in the four key phases of project management enable a project team to:

- Link project goals and objectives to stakeholder needs.
- Focus on customer needs.
- Build high-performance project teams.
- Work across functional boundaries.
- Develop work breakdown structures.
- Estimate project costs and schedules.
- Meet time constraints.

- Calculate risks.
- Establish a dependable project control and monitoring system.

Project management is a challenging task with many complex responsibilities. Fortunately, there are many tools available to assist with accomplishing the tasks and executing the responsibilities. Some require a computer with supporting software, while others can be used manually. Project managers should choose a project management tool that best suits their management style. No one tool addresses all project management needs. Program Evaluation Review Technique (PERT) and Gantt Charts are two of the most commonly used project management tools and are described below. Both of these project management tools can be produced manually or with commercially available project management software.

Some of those tools used by project managers in all domains are;

***Project Plan*:** All the projects that should be managed by a project manager should have a project plan. The project plan details many aspects of the project to be executed.

***Milestone Checklist*:** This is one of the best tools the project manager can use to determine whether he or she is on track in terms of the project progress.

***Gantt Chart*:** Gantt chart illustrates the project schedule and shows the project manager the interdependencies of each activity. Gantt charts are universally used for any type of project from construction to software development.

Project Management Software: With the introduction of computer technology, there have been a number of software tools specifically developed for project management purpose. MS Project is one such tool that has won the hearts of project managers all over the world.

***Project Reviews*:** A comprehensive project review mechanism is a great tool for project management. More mature companies tend to have more strict and comprehensive project reviews as opposed to basic ones done by smaller organizations.

***Delivery Reviews*:** Delivery reviews make sure that the deliveries made by the project team meet the customer requirements and adhere to the general guidelines of quality.

***Score Cards*:** When it comes to performance of the project team, a scorecard is the way of tracking it. Every project manager is responsible of accessing the performance of the team members and reporting it to the upper management and HR.

Conclusion: A project manager cannot execute his / her job without a proper set of tools. These tools do not have to be renowned software or something, but it can pretty well be simple and proven techniques to manage project work.

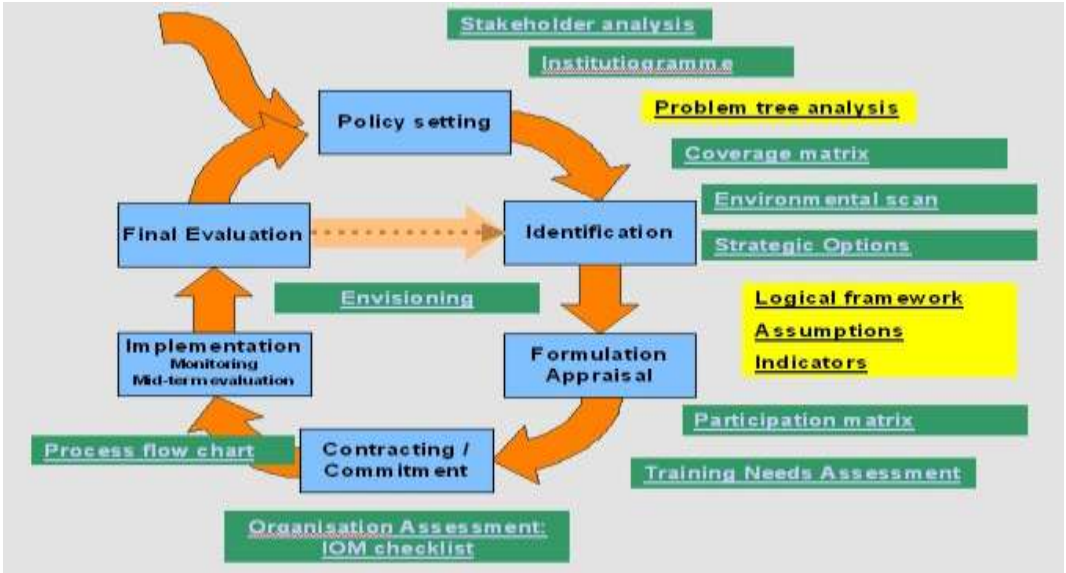


Fig : 2 Project management tools

3.3 Risk Management Tool

Almost all organizations have experimented with remotely located development facilities (DSD) for seeking lower costs and skilled resources. This is because of the several factors those have contributed to built this scenario; such as the business market proximity advantages including the knowledge of customers and local conditions, pressure to improve time-to-market by using time zone differences in "round the clock" development and even the need to have a global resource pool to successfully and cost competitively have resources, wherever located.

Risk management is the systematic application of management policies, procedures and practices to the basis of identifying, analyzing, evaluating, treating and monitoring risk. On top of the regular risk repositories and check lists, several specific risks must be stressed in global development projects. They relate to two major drivers such as insufficient processes and inadequate management.

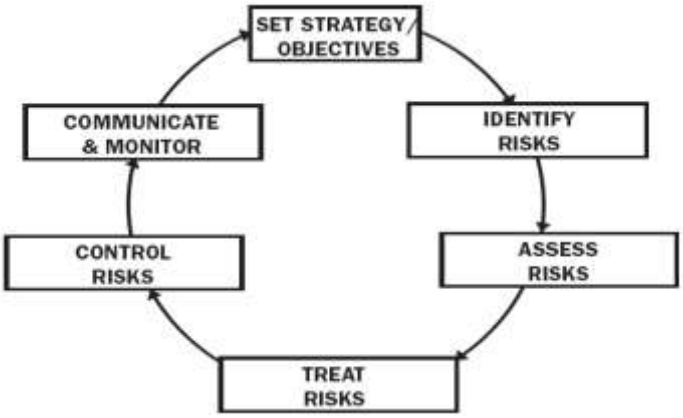


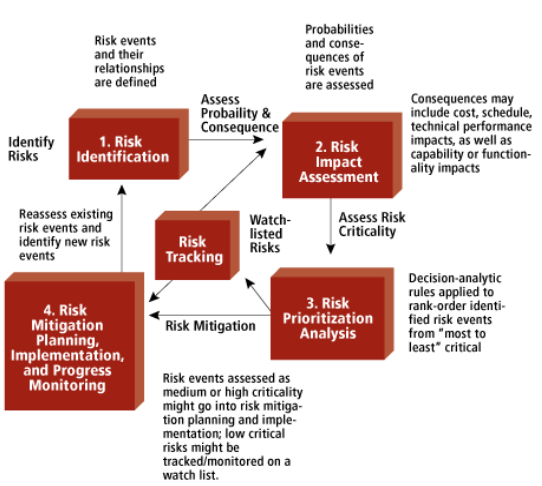
Fig: 3 Risk Management process

The Risk Analysis Tool (RAT) provides a method for consistent and coherent identification of risk elements. It also allows users to effectively prioritise actions designed to reduce the effect of those elements.
The RAT tool has evolved over time to be a sophisticated yet simple program for quantifying the level of risk present in any air incident. Requiring only a brief series of program inputs to produce a valid result, the tool expresses the relationship between actions and consequences and provides a quantifiable value to these relationships.

Selecting the Right Tool: It is important that the organization defines the risk analysis and management process before selecting a tool. Ultimately, the tool must support the process.
The3se are the criteria to consider when selecting a risk analysis and management tool
Aligned to risk analysis objectives—does the tool support the analysis that the organization is trying to accomplish? Is the organization attempting to implement an ongoing risk management process or conduct a one-time risk analysis?

Supports decision making—does the tool provide the necessary information to support decision making?
Accessibility—Is the tool accessible to all users and key stakeholders? Can the tool be located or hosted? where all necessary personnel can access it?
Availability of data—Is data available for the tool's analysis?
Level of detail—Is the tool detailed enough to support decision making?
Integration with other program management/system engineering processes—Does the tool support integration with other program management/system engineering processes?

Many tools are available that support the implementation of program risk management. Many tools also can be used to support the management of project, enterprise, and system-of-systems risks. Some of the commercial Tools like; Risk Radar and Risk Radar Enterprise—American Systems, Active Risk Manager—Strategic Thought Group, Contractor Tools and customized Tools etc.



Likelihood	Consequences				
	Insignificant (Minor problem easily handled by normal day to day processes)	Minor (Some disruption possible, e.g. damage equal to $500k)	Moderate (Significant time/resources required, e.g. damage equal to $1million)	Major (Operations severely damaged, e.g. damage equal to $10 million)	Catastrophic (Business survival is at risk damage equal to $25 Million)
Almost certain (e.g. >90% chance)	High	High	Extreme	Extreme	Extreme
Likely (e.g. between 50% and 90% chance)	Moderate	High	High	Extreme	Extreme
Moderate (e.g. between 10% and 50% chance)	Low	Moderate	High	Extreme	Extreme
Unlikely (e.g. between 3% and 10% chance)	Low	Low	Moderate	High	Extreme
Rare (e.g. <3% chance)	Low	Low	Moderate	High	High

Fig.4: Analytical view of risk management

3.4 Requirements management tools

Requirements management is a systematic approach to finding, documenting, organizing, and tracking the requirements of a system. Effective requirements management improves the communication of project goals, enhances collaborative development, reduces project risk, and increases the quality of your product. Requirements management includes the following tasks:

- Adopting a requirements management plan to define the process, team roles, artifacts, and more
- Analyzing the problem
- Gathering and examining stakeholder needs
- Defining the system through a vision document, use-case model, glossary, and other artifacts
- Refining the definition of the system in detailed requirements, use cases, and documents
- Using traceability to ensure coverage of feature requirements by use cases, functional requirements, or supplementary requirements
- Managing the scope of the system through requirement attributes, documents
- Managing requirement change with traceability and requirement attributes

Requirements management is a capability to which a project outcome (product or service) should confirm. Broadly the requirement activities are; *Investigation, Feasibility, Design Construct and test and Release.*

There are a large number of requirements tools available and these tools help to capture, develop and manage *structured* requirements in a collaborative fashion. Good requirements tools can offer several benefits to the requirements management process.

Fig. 5: Use of CASE tools in RM

Enterprise-level **Requirements Management Tools:** These tools are primarily targeted at large, enterprise-level implementations. They are usually very expensive but are also loaded with features for enterprises.
Ex. IBM Rational DOORS Borland Caliber

Mid-market Requirements Management Tools: These provide a nice balance between the feature set of "enterprise-level" tools listed above and ease-of-use of "entry-level" tools listed below.
Ex. Accompa , Jama
Entry-level **Requirements Management Tools:** These tools are affordable and can be used to manage requirements in a structured fashion – especially at smaller organizations. There is one caveat – these are *not* focused on requirements, but rather on issues/bugs. Ex. Atlassian JIRA , FogBugz

3.5 Metrics tools

Software process and product metrics are quantitative measures that enable software people to gain insight into the efficacy of the software process and the projects that are conducted using the process as a framework. Basic quality and productivity data are collected. These data are then analyzed, compared against past averages, and assessed to determine whether quality and productivity improvements have occurred. Metrics are also used to pinpoint problem areas so that remedies can be developed and the software process can be improved. Software analysis generally extracts arbitrary properties of software source code. General or custom analyses of software can be implemented using DMS.

Software metrics are a special kind of analysis focused on the structure of the source code. Classic software metrics range in variety from the very simple Source Lines of Code (SLOC) to more complex measures such as Cyclomatic Complexity measurements. Typical metrics report provide details on individual modules and summaries for subsystems. Such metrics are widely used to judge the quality of source code, enabling a software organization to more effectively focus its attention on the lower-quality portions of their portfolio. A typical SD metrics report can be seen for Java.

The advantages of classical Metrics tools are:

1. Wide acceptance of basic value of metrics
2. Unbiased assessment of source code quality
3. Repeatability of measurements
4. Ease of measurement
5. Ability to judge progress in enhancing quality by comparing before and after assessments

Software metrics are necessarily computed on the structure of the source code. This means metrics must be extracted from a parse of the program text. All of SD's Metrics tools use DMS's ability to parse large scale software systems, and are based on the language definition modules used to drive DMS for large scale software reengineering tasks. SD's metrics tools are presently available on Windows 7, Vista, XP, and 2000. Few available metric tools are: SD offers a family of language-specific metrics tools based on DMS:

C# (1.2, 2.0, 3.0, and 4.0), COBOL (IBMEnterprise) , Java, Logix5000 (IEVER 2.6) and VBScript (Beta)

Each of the language-specific metrics tools provides standard metrics down to the level of methods or procedures, and summary rollups of larger units such as classes, files and (directory) subsystems.

SD also provides basic metrics for a much wider variety of languages, via the Source Code Search Engine. These metrics are computed at the file and directory subsystem level, across multiple languages. This is effective for obtaining metrics across very large application systems.

Some other specific tools are for detecting the amount of redundant code in source code (high redundancy implies poor quality, hard-to-maintain code). Similarly, SD can provide tools and services related to detecting dead code.

Semantic Designs can build custom Metrics tools for: Unusual languages or dialects, Different metric models, Metrics across language boundaries. Every organization can also build its own custom tools based on DMS.

Virtually all metrics tools (and most reverse engineering tools in general) are limited to simply reporting. SD's DMS technology can be brought to bear to automate, within limits, activities to enhance software quality defined by metrics. As an example, the CloneDR service can not only report on duplicate code, but can actually remove certain types automatically. Inquire about custom services for enhancing software quality

3.6 Modelling tools

Models are essential in software engineering. A model is an abstract representation of an object. We might model the decomposition of a system into components and their dependencies. A model can demonstrate the consistency of the system specifications or be a predictor of system behavior. The analysis of system performance in data throughput or computation efficiency so as to meet critical real-time performance requirements depends on modeling that aspect of system behavior.

A UML tool or UML modeling tool is a software application that supports some or all of the notation and semantics associated with the Unified Modeling Language (UML), which is the industry standard general purpose modeling language for software engineering.UML tool is used broadly here to include application programs which are not exclusively focused on UML, but which support some functions of the Unified Modeling Language, either as an add-on, as a component or as a part of their overall functionality.

The various steps are...*Diagramming, Round-trip engineering, Code generation, Reverse engineering, Model and Diagram Interchange and Model Transformation.* Example of few tools:

Tools	Vendor	Platform
AgileJ StructureViews	AgileJ	Cross-platform (Java)
Altova UModel	Altova	Microsoft Windows
ArgoUML	Tigris.org	Cross-platform (Java)
astah*	Change Vision, Inc.	Multi-platform
ATL	Obeo, INRIAFree software community	Cross-platform (Java
Borland Together	Borland	Cross-platform (Java)

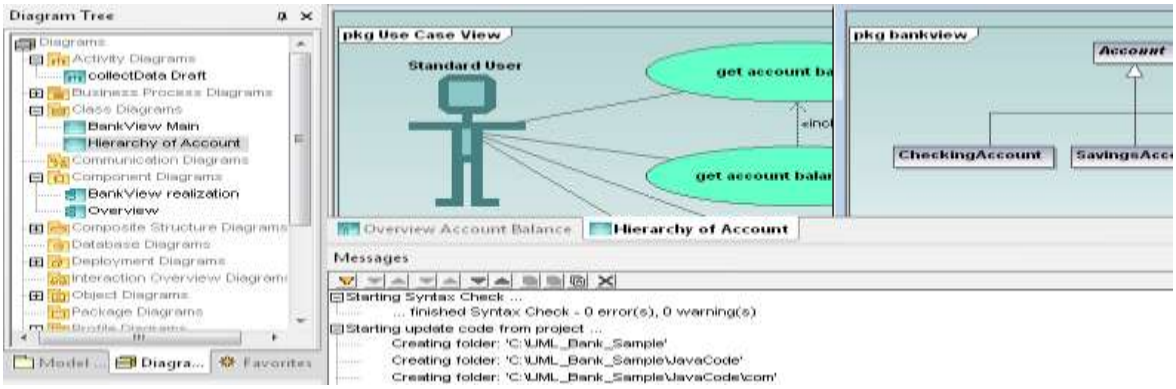


Fig.6; Alternate figure for the use of modeling tool

3.7 Programming tools

A programming tool or software development tool is a program or application that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object. Programming tool or programming software is a sub-category of system software but sometimes it is stated as a separate category of software along with application and system software.

Software tools come in many forms:

***Binary compatibility analysis*:** icheck, ABI Compliance Checker
***Bug Databases*:** Comparison of issue tracking systems - Including bug tracking systems
***Build Tools*:** Build automation, List of build automation software
Code coverage: Code coverage#Software code coverage tools. Software Diagnostics
***Code Sharing Sites*:** Freshmeat, Krugle, Sourceforge, GitHub. See also Code search engines.
***Compilation and linking tools*:** GNU toolchain, gcc, Microsoft Visual Studio, CodeWarrior, Xcode, ICC
Debuggers: *Debugger*#List of debuggers. See also Debugging.
***Development Productivity Tools*:** JRebel eliminates the build and redeploy phases of Java EE Development by mapping the project workspace directly to any type application server in real-time
***Disassembles*:** Generally reverse-engineering tools.
***Documentation generators*:** Comparison of documentation generators, help2man, Plain Old Documentation, Asciidoc.
***Formal methods*:** Mathematically-based techniques for specification, development and verification *GUI interface generators*
***Library interface generators*:** SWIG *Integration Tools*
***Parser generators*:** Parsing#Parser development software
***Performance analysis or profiling*:** List of performance analysis tool *Refactoring Browser*
Revision control: List of revision control software, Comparison of revision control software
***Scripting languages*:** PHP, Awk, Perl, Python, REXX, Ruby, Shell, Tcl

***Search*:** grep, find
***Source code Clones/Duplications Finding*:** Duplicate code#Tools
***Source code formatting*:** indent
***Source code generation tools*:** Automatic programming#Implementations
***Static code analysis*:** List of tools for static code analysis
***Text editors*:** List of text editors, Comparison of text editors
***Unit testing*:** List of unit testing frameworks
Integrated Development Environments combine the features of many tools into one package. They for example make it easier to do specific tasks, such as searching for content only in files in a particular project. IDEs may for example be used for development of enterprise-level applications.
Different aspects of IDEs for specific programming languages can be found in this comparison of integrated development environments.

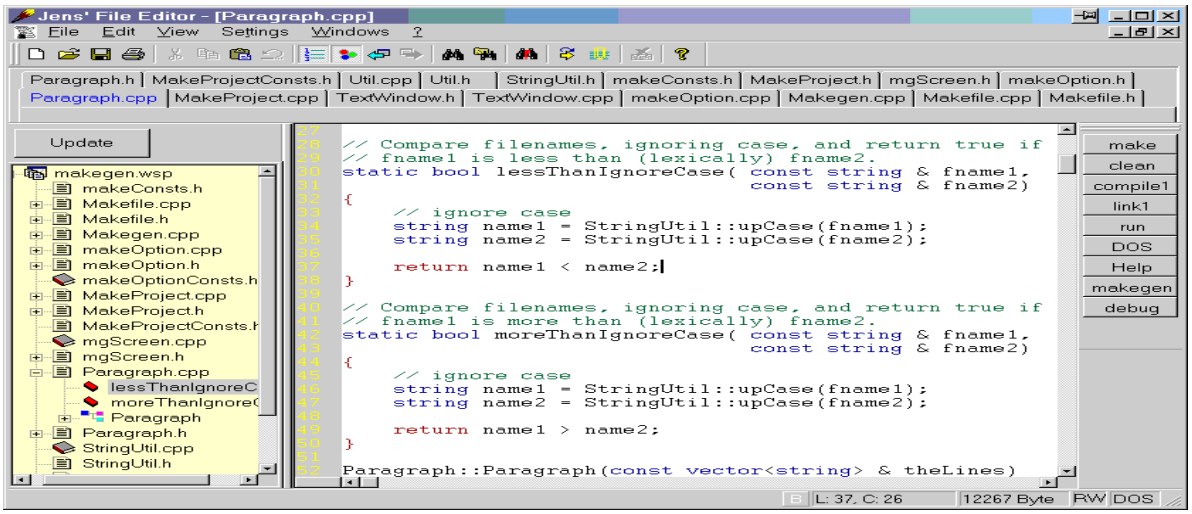


Fig. 7: A programming tool

3.8 Interface design tools

Interface design deals with the process of developing a method for two (or more) modules in a system to connect and communicate. These modules can apply to hardware, software or the interface between a user and a machine. An example of a user interface could include a GUI, a control panel for a nuclear power plant, or even the cockpit of an aircraft.

In systems engineering, all the inputs and outputs of a system, subsystem, and its components are listed in an interface control document often as part of the requirements of the engineering project.User interface design or user interface engineering is the design of computers, appliances, machines, mobile communication devices, software applications, and websites with the focus on the user's experience and interaction. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often called user-cantered design. Good user interface design facilitates finishing the task at hand without drawing unnecessary attention to itself. Graphic design may be utilized to support its usability. The design process must balance technical functionality and visual elements (e.g., mental model) to create a system that is not only operational but also usable and adaptable to changing user needs.

User Interface Design Tools: ForeUI is an easy-to-use UI prototyping tool, designed to create mock-up / wireframe / prototypes for any application or website you have in mind. With ForeUI, your prototype project will be sinkable. Easily change the look and feel of your prototype by simply switching the UI theme.
Example-.FORE UI, LIVEPIPE UI, DESIGNER VISTA

Fig.8: Prototyping tools for UI

3.9 Test management tools

Test management tools give teams the ability to consolidate and structure the test process using one test management tool, instead of installing multiple applications that are designed to manage only one step of the process. Test management tools allow teams to manage test case environments, automated tests, defects and project tasks. Some applications include advanced dashboards and detailed tracking of key metrics, allowing for easy tracking of progress and bug management.

***Implementation*:** A test management tool that includes everything needed to manage the test process can save testers the hassle of installing separate applications that are necessary for the testing process. They can be implemented with minimal programming ability, allowing for easy installation and monitoring of the test process across multiple project groups. Once installed, teams have instant access to a user interface and can immediately start running and recording test cases. These types of applications are designed to simplify the test management process with high levels of automation and tracking built in, yet don't require advanced programming skills or knowledge to implement. They are useful for teams who manage a variety of test cases and for larger teams who need an all-inclusive application for project management.

***Uses*:** Once a project has kicked off, a test management tool tracks bug status, defects and projects tasks, and allows for collaboration across the team. When administering test cases, users can access a variety of dashboards to gain access to data instantly, making the test process efficient and accurate. The type of dashboard used is determined by the scope of the project and the information and data that needs to be extracted during the testing process. Data can be shared and accessed across multiple project teams, allowing for effective communication and collaboration throughout the testing process.

Examples of few tools. 1. Aqua 2012 by Andagon, 2. Enterprise Testerby Catch Software and 3. PractiTest by PractiTest..etc. Many such tools are available, from simple diagrammatic tools to fully blown integrated environments with project management, document repository and code generation capabilities.

Fig. 9: UI execution environment

IV. Integration Of Case Tools For Dsd

CASE technology includes both individual CASE tools and integrated CASE environments. There are two nonexclusive approaches to improving software quality and productivity, a micro-management approach and a macro-management approach (Sage and Palmer, 1990). A micro-management approach attempts to achieve incremental improvements in the various phases of a software development process but leaves the overall process unchanged. A macro-management approach is systemic and wholistic, attempting to achieve improvements by addressing the software development process in its entirety. Implementing individual CASE tools is a micro-management approach that frequently fails to achieve the desired results. Implementing integrated CASE environments, such as a Factory Support Environment, is a macro-management approach. Integration is key to achieving significant improvements in quality and productivity. There are four important forms of integration required to create a fully integrated CASE environment - management, process, team and tool (Bell and Sharon, 1995). Management integration refers to using CASE tools to control and monitor software development. The tools should collect cost, productivity and quality metrics to allow management of schedules and budgets and facilitate improvement initiatives. Process integration refers to using CASE tools that collaborate specifically to support the process and span the process life cycle. The process should not only identify the phases and tasks of software development but it should also identify the tools for each task and the sequence of their use. The tools should allow software developers to follow the process effectively and provide for configuration control. Team integration, or organizational integration refers to integrating the Factory Support Environment and the organization. For example, tools can be used to electronically network software development teams. The tools should provide for communication and shared access to data while maintaining data integrity. Team integration provides for the "people centered view" of a software factory. Tool integration refers to using CASE tools that share user-interfaces, data and functionality. The next section discusses tool integration in more detail. The essential capabilities of an integrated CASE environment needed to capture the four forms of integration are process management, project management, requirements management, configuration management, document management, a repository, and project verification and validation. In addition the environment should be flexible, extendible and capable of supporting an organization working on multiple projects(Ref. 05).

CASE Tool Integration Of the four forms of integration, tool integration has been the subject of greatest focus. Tool integration addresses the mechanisms of linking individual CASE tools together. There are three dimensions of tool integration - user-interface or presentation integration, data integration and control integration. User-interface integration refers to using a common look and feel among various CASE tools to facilitate ease of use and quick learning. Data integration refers to the representation, conversion and exchange of data in a common standard. It determines to what degree data generated by one tool can be accessed and understood by another. Control integration refers to tool invocation, shared functionality, and the ease of communications between tools. Data and control integration are closely related because tools share functionality by exchanging control messages that contain data and data references. Tools are called interoperable when full data and control integration exists between them(Ref. 06).

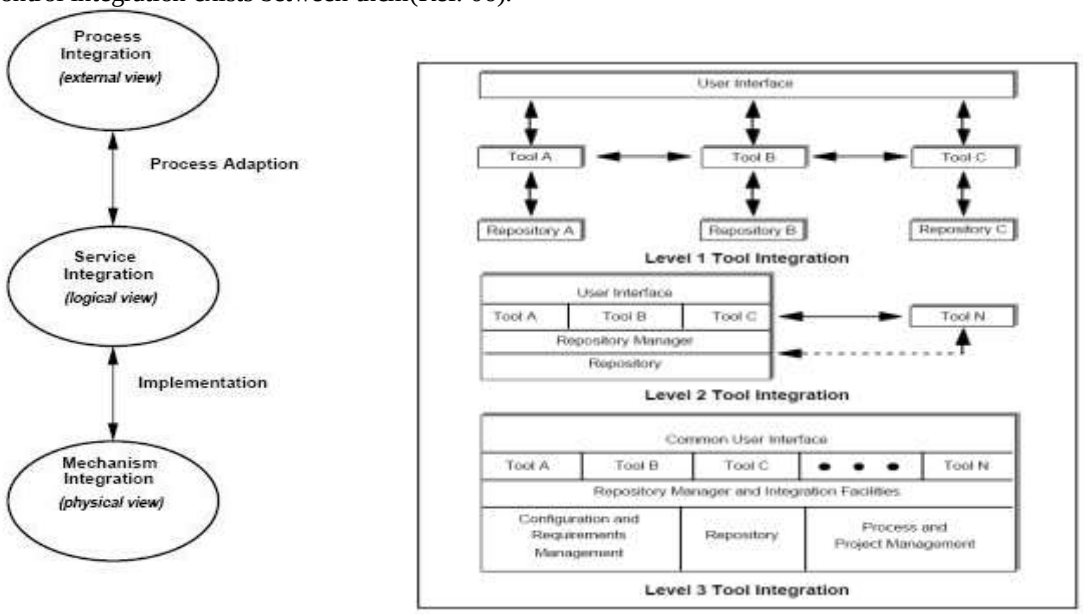


Fig.10 (a) : Schematic layers of integration (b): Levels of CASE tools integration

Figure 10. (a) shows three levels of tool integration (Bell and Sharon, 1995). Which level is achieved depends on how the tools were developed. At level 1, individually developed tools are used. They are likely to have common user interfaces and data import/export formats but are unable to share a single repository. At level 2, tools developed together as a suite are used. They are tightly integrated and optimized amongst themselves but their integration with other tools remains at a level 1 capability. At level 3, tools developed to meet formal interoperability standards are used. These tools may have been developed individually but together they are capable of forming a highly integrated environment (shown below).

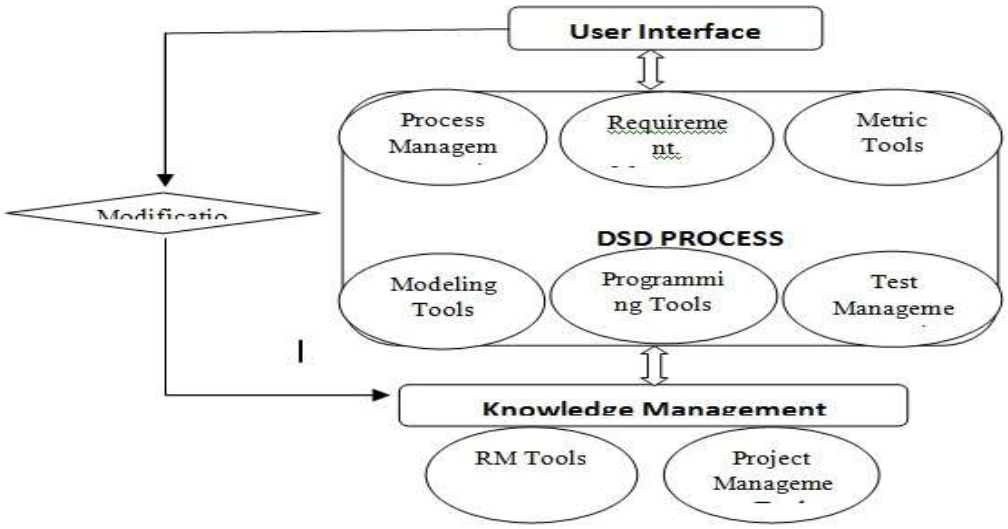


Fig.11: Prototype framework for CASE tools integration in DSD management

V. Conclusion

Distributed software development process is becoming more complex day-by-day because of the significant variation in software products and to manage the DSD process, we have proposed integrated environment for distributed software development process through a set of different CASE tools component engineering. This includes tools to support process management and local tool integration and task automation, flexible collaborative editing, distributed work coordination, and a distributed component repository. Some 3rd party tools have been integrated into this environment, including a programming environment and distributed file repository. These tools can all be used over the internet to support distributed software development. Most utilize a decentralized software architecture, which is fault-tolerant and provides good performance over a range of internet communication technologies. Further enhancements to this environment include the addition of distributed testing and monitoring tool support, distributed user interface development tools, and further work codifying the processes, notations and implementation and aimed at providing tool developers with a cleaner, more extensible architecture on which to build and integrate tools and tool services.

References

[1] S. Larsen, "Evaluation of Software Modeling Tools for Agile and Industrial Process," Dept. Computer and Information Science, Norwegian Univ. Science and Technology, 2006; www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2006/larsen-fordyp06.pdf.

[2] J. Portillo Rodríguez, C. Ebert, and A. Vizcaíno, "Tools and Technologies for Distributed Teams," IEEE Software, vol. 27, no. 4, 2010, pp. 10–14.

[3] C. Ebert, Global Software Engineering: Distributed Development, Outsourcing, and Supplier Management, Wiley/IEEE CS Press, to be published in 2011.

[4] "Software & Systems Process Engineering Meta-Model Specification (SPEM), Version 2.0," Object Management Group, 2008; www.omg.org/spec/SPEM/2.0/PDF.

[5] "Business Process Model and Notation (BPMN), Version 2.0, Beta 2," Object Management Group, 2010; www.omg.org/cgi-bin/doc?dtc/10-06-04.pdf.

[6] V. Nikulsins and O. Nikiforova, "Tool Integration to Support SPEM Transformations in Eclipse," Scientific J. Riga Tech. Univ., vol. 45, 2010, pp. 60–67; https://ortus.rtu.lv/science/en/publications/8640/fulltext.