TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Improving the Usability of Protégé Plug-In for Artifact Management using Taxonomic Paths

Chenreddy pradeepreddy

MASTER THESIS 2009
COMPUTER ENGINEERING

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Förbättring av användbarhet av Protégé Plug-in Artifact Manager

Improving the Usability of Protégé Plug-In for Artifact Management using Taxonomic Paths

Chenreddy pradeepreddy

Detta examensarbete är utfört vid Tekniska Högskolan i Jönköping inom ämnesområdet datateknik. Arbetet är ett led i teknologie magisterutbildningen med inriktning informationsteknik. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Kurt Sandkuhl
Examinator: Vladimir Tarasov

Omfattning: 20 poäng (D-nivå)

Datum: 2009-05-08

Arkiveringsnummer

Postadress:	Besöksadress:	Telefon:
Box 1026	Gjuterigatan 5	036-10 10 00 (vx)
551 11 Jönköping		

Abstract

The goal of this thesis is to improve Usability and functionality of a tool for artifact management, which applies taxonomic paths for categorizing artifacts. The main issues of using the taxonomic paths are used for categorization and should improve the precision when retrieving documents. The results show the improvements in functionality and usability of the artifact manager. This thesis explains about Usability, re-engineering, and necessary infrastructure to improve the performance of the artifact manager tool.At the end of the thesis necessary modifications has been done to improve usability and functionality of artifact manager.

Sammanfattning

Målet med denna uppsats är att förbättra användbarheten och funktionaliteten av ett verktyg för handhavande av artefakter. Verktyget använder sig av taxonomiska sökvägar för kategorisering. Huvudanledningen till att använda taxonomiska sökvägar är att träffsäkerheten vid dokumentsökningar bör öka vid kategorisering.Resultatetdelen visar förbättringarna i funktionalitet och användbarhet för verktyget.Uppsatsen är inriktad på användbarhet, nytänkande och nödvändig infrastruktur för att öka prestandan på verktyget. I slutet av uppsatsen har nödvändiga ändringar gjorts för att öka användbarheten och funktionaliteten på artefakthanteringsverktyget

Acknowledgements

I find myself overwhelmed in offering my parents all my thanks in dedicating this thesis to them.

First of all I would like to thank Kurt Sandkuhl for giving me the opportunity to work with this thesis. I would like to thank to my friends nanda, debasis and pranaya who has been a great support all through this work. Also I would like to thank my parents Dharma reddy and Sukanya for their patience, moral support and understanding allowed me to complete this thesis.

Thanks to all my friends who were far away from me, still providing me the confidence, encouragement throughout the thesis.

To each of the above I extend my deepest appreciation. Thank you for all your support and guidance.

Key words

Protégé, Ontology, Plug-in, Artifact.

Contents

List of Figures

List of Abbreviations

EO = Enterprise Ontology.

1. Introduction

Many engineering disciplines heavily use documents to capture requirements, specifications, design decisions, assembly instructions, test procedures or other artifacts contributing to development and design processes. Although there is a clear trend towards model-based development, i.e. using model-based representations of all artifacts in an integrated tool chain, reality in a lot of enterprises continues to be characterized by document management, document retrieval and the struggle for keeping related documents consistent. A document in this context denotes a structured amount of information that is meant for human perception. An artifact is the general term for all work products in an engineering process, many of them being represented as documents. Support for managing documents has been subject of research since decades in fields like document engineering. Progress in ontology engineering creates new possibilities for easing document management by using [1] ontology's.

The subject of this thesis is to improve usability and functionality of a tool for artifact management, which applies taxonomic paths for categorizing artifacts. A taxonomic path is a sequence of nodes connected by the "is a" relationship within a taxonomy. Using an individual set of such paths for a specific artifact is supposed to enhance the meta-data for the artifact under consideration, i.e. the taxonomic paths are used for categorization and should improve the precision when retrieving [2] documents.

This report describes our thesis that is a part of master education information technology at Jonkoping University.

1.1 Background

The primary task of the thesis is improving the usability of the existing Artifact Manager plug-in for Protégé by evaluating the usability of the current system, identifying improvements regarding the user interface of the existing plug-in, identifying improvements of the existing functionality, including new features and implementing the user interface improvements and functionality improvements. The implementation work includes the necessity to migrate to the next Protégé version.

The theoretical part of the work focuses on the usability aspects and their consequences with respect to the architecture and implementation of the plug-in and re-engineering concepts.

1.2 Purpose

The main purpose of the thesis is to enhance the existing prototype of an artefact manager application in order to show that it is possible to use ontology to specify artefacts and look for them by considering the enterprise ontology. Artefacts describe documents related to products or processes for example. In order to get efficient Searching results we applied threshold technique to filter the data. Other techniques can be used together with the ontology part together to facilitate the main goal which is to allow faster and cleverer management and to find the existing artifacts for the engineer.

Theoretical research is focused on the usability aspects and their consequences with respect to the architecture and implementation of the plug-in and re-engineering concepts. Basically the theoretical part needs comprehensive research and knowledge of usability heuristics. Nielsen has described "*Heuristic evaluation is the most popular of the usability inspection methods. Heuristic evaluation is done as a systematic inspection of a user interface design for usability. The goal of heuristic evaluation is to find the usability problems in the design so that they can be attended to as part of an iterative design process"* [4], based on the above theory we have done intense study on the existing artifact manger tool and came up with the suggestion to improve the usability. The achieved improved usability has been discussed clearly and after rigorous study of system re-engineering [3] concept we have tried to imply the system re-engineering concept to the existing artifact manger to improve the functionality of the tool. Finally we have reached to a conclusion that the usability heuristics can be interface with the software re-engineering concept to improve the usability and functionality of existing artifact manger.

The application should be based on an existing ontology editor protégé and be able to manage artifact types and artifacts. Browsing and selecting/highlighting sub-graphs from the enterprise ontology are other requirements for the application.

1.3 Limitations

To begin our work, we had to finalise the boundaries and limitations of the thesis. To get successful results it is vital to select proper methodology to execute the project. Few limitations of this master thesis as described below.

- Enhancement to this software should not affect the basic functionality of existing plug-in.
- Our application does not support all versions of ontology.
- It is specific software created for the protégé tab-widget plug-in.

1.4 Outline

The introduction describes our task and how this thesis was set up. It is followed by descriptions of different usability heuristics and reengineering technique that are used for the thesis. In the implementation part identifying improvements and functionality of user interface of the existing plug-in are explained.
The modified user interface, performance and functionality of the artifact manager are then presented ad analyzed. Modified artifact manager is our resulting product. The applied usability heuristics research outcome is also described in this chapter.
Finally .in the conclusion and discussions chapters, we summarize what has been achieved and discuss the results.

2 .Theoretical Background

Basically our task is to improve the performance and usability of the existing artefact manager tool for protégé. For that reason to increase the performance of the tool we have applied software reengineering concept and as well as to increase the usability of the artefact manager we have used usability heuristics.

In the first section we have discussed ten usability heuristics precisely and later we have explained core concept of software reengineering.

2.1. Usability Heuristics

According to Jacob Nielsen [4] there are ten Usability heuristics, they are called "heuristics" because they are more in the nature of rules of thumb than specific usability guidelines. In the below sections we have elaborated the ten usability heuristics which would be applicable to develop this project to greater extent.

2.1.1. Visibility of system status

Interaction between system and user is a vital process. This increase co-ordination and access level between user and system. This can be achieved by giving frequent information to the user about ongoing processes from system side with in realistic time. [4].

Provide appropriate feedback

It is vital for a user friendly tool to provide accurate and precise feedback within a stipulated time. System interacts with the user by providing an accurate progress indicator that displays the status of a task and if this status is inappropriate, it will drive the user towards the credibility of progress indicators thus lead to a conclusion that environment is less comprehensible and well-suited. When the system displays a standard error message indicating a logical or technical malfunction and does not provide any guidance to diagnose the problem, this might hamper the running task [5].

Every time user provides an input or performs an action, there must be a prompt indication that application has received the user's input. A good tool shall provide the user with the current status of action. System to be designed in such a way that the execution of the command is known to user and also in case of execution failure proper pop up message must be displayed. Tool can be made more user friendly by providing appropriate status and guide the user to complete execution during failure. For example when you are installing an application and installation got aborted because of low memory; the user may abort the installation due to unavailability of the solution [5].

A good warning or error message should contain the following elements:
1. Show the description of the problem
2. A sequence of alert messages to guide the user to find a solution for the problem.
Both of these elements should be presented in simple, non-technical and jargon-free language [5].

Keep the User Informed

A well designed system will provide the user with an approved status of the application at an exact time using appropriate feedback. The tool should avoid speculative work for the user regarding the status of the system or application [5].

For lengthy executions, tool should display a progress indicator that can provide timing information about how long the process will take to complete. Users don't need to know exactly in seconds about the ongoing process, but an approximation time is helpful, for example when we query for the price of a product with a combination of specifications for which the application takes more time to respond and there is no estimation about the result of the query this might make the user exasperated and avoid its usage [5].

Even though extremely responsive applications can differ widely from one another, they share the following characteristics [5].

- System should provide instant feedback to users, even when it cannot complete their needs instantly.
- System should give sufficient feedback for users to recognize what they are doing, and manage feedback according to user's abilities to understand and reply to it.
- System should inform when processing is in progress
- System should provide approximation time to finish the ongoing process.

2.1.2. Match between system and the real world

The system should design in such a way that the bipolar communication between user and system must be realistic and user friendly, system must understand and communicate with language, phrase, word, logics, conventions, and feedback which would be easy to understand to the end user [4].

An application developer uses terms that make technical or logical sense to him during his test and debugging phases for displaying messages at various stages of the application and errors. This is bad practice since the user might not be familiar to these terms. For example if the programmer uses STDOUT as one of the display messages when you enter an application, STDOUT is not understood by all the users, this is the case when

user cannot understand the status of your application where more meaningful message is required to say that you have opened the application[7].

There are few cases where users cannot read the programmers' mind, they need visual clue to guide them through out the program, and these clues often make use of real world conventions to be effective. For example if you provide '+' sign to perform intersect operation, user will be driven to a different set of results [7].

The "match between system and real world" means that the system should go after real-world conventions as strictly as possible, to allow the user to understand how to use the program. Real-world representations and natural interactions give the interface a familiar look and feel and can make it more intuitive to learn and use. If you use a tree picture for CD/DVD drive, user will not be able to understand that this is the location to browse for data of CD/DVD drive.

We can often take advantage of users' knowledge of the real world by using metaphor that is, a well-known concept from the outside world to represent elements within your application. For example when you are generating a report based on time, a picture of clock in the first page of report gives an indication to the user about the report being based on time. Another simple example can be:

- A file folder icon suggests that it is the location where documents are placed.
- A trash can icon informs the user that discarded files can be found here.
- While using metaphors, it is important to neither take the metaphor too factually, nor to extend the metaphor beyond its realistic use.

There are few factors to consider when using a metaphor:

Once you decided to use metaphor, its usage should be same throughout the interface, rather using once at a specific point. Even superior would be to use the same metaphor extend over numerous applications.fro example we can find floppy symbol in MS Word to save and in MS excel if floppy symbol is used for opening a file from the floppy drive, creates non-uniformity of the metaphor causing confusion in the minds of the user[6].

Metaphor is not always essential. In most of the cases the normal purpose of the software itself is easier to understand than any real-world analogy of it. Do not damage a metaphor in adapting it to the program's real purpose. Nor should you damage the sense of a particular program feature in order to adapt it to a metaphor [6].

2.1.3. User control and freedom

Some time Users choose system functions by error and will require a visibly marked "emergency exit" to leave the surplus state without having to go through a comprehensive dialogue. The system should supports undo and redo actions [4].

As much as possible, allow users to do whatever they want at all times. Users should always feel in control, able to do what they want when they want. Users should be able to switch between different tasks at any time. Avoid using interfaces that lock them into one operation and prevent them from switching to anything else until that operation is completed. Users should always have a clear path out. Avoid interfaces that make users feel trapped. Use constant illustration elements to enable people to navigate fast but also to allow them have reliable landmarks, giving people a sense of control on the navigation [5].

To give users control over the system, allow them to achieve tasks using any cycle of steps that they would naturally use. Don't limit them by artificially restricting their choices to your notion of the "correct" sequence. The user must also be capable to tailor aspects of their environment to fit individual preferences. It is very important, nevertheless, to keep away from the ambush of allowing too much configuration, or allowing the configuration of parameters that mainly accepted by user [5].

It is very important to provide end-users with the capabilities they need while helping them avoid hazardous, irreversible actions. For example, in situations where the user might demolish data accidentally, in this case it is always better to provide a warning message before proceed for final deletion [5].

2.1.4. Consistency and standards

There should be consistency in design level of the system, so that user will wonder with the flow, action, and situation of the system. Consistent design approach must be followed while developing a system [4].

Consistency
Idea behind building any system is to have a consistent performance of the tool. When the consistency of the system fails attributes like usability, navigation**, Performance, user-friendliness** does not facilitate the user in using the tool. Consistency plays a vital role in user interface. Some things to keep in mind for a consistent tool:

- Layout of application be with nominal eccentricity
- Navigation be apparent and consistent and not every click a venture

Usability
Usability is one of the trivial features that makes the user chose a particular system/tool. Due to resources venture towards Usability is bounced and reason behind this being a simple one that is many errors could be discovered at the initial stage of the system/tool development. There could be many meetings to understand the requirement of the user, deciding the number of meetings to gather the requirement depends on the tool/system

Navigation
Navigation is to an application as table of contents are to a book. Navigation in an application has to be clear and simple for the user. With a glimpse on the layout user should be able to identify which segment of the application is he accessing and where the user is being headed to. Navigation should ascertain the user about: Which segment of the application is he in? Usually application/tool designers tend to have aspect of projecting designer's preferences onto a design. Nomenclature used for an application has to be a part of navigation of your application.

Perfor**mance**
We live in 3rd generation mobile world, where at the press of a button of your cell phone you get directions to unknown destinations with GPS in it, at such scenario we cannot wait for the download to be completed while we drive from office to a friend's place. A pragmatic standards-based design helps in achieving the need. With simple, semantic markup and intangible presentation of the structure we achieve the desired performance concern

2.1.5. Error prevention

A watchful design will be more helpful to avoid a trouble from occurring in the preliminary stage than a good error messages. Inspection the errors and eliminating them is the best way before confirming the option to the users [4].

One of the methods to help users make true choices is to predict frequent troubles and providing feedback and communicate at every stage. Designer has the load to keep the user out of difficulty. When requested the interface has to provide visual cues, reminders, list of choices, and other aids. Recognition is easier for humans than recall. Supplemental support is provided by appropriate and hover help, as well as agents. An opportunity to eliminate user mistake and confusion is the gist of it [5].

The difficult task is to write helpful error messages which user understands. Displaying error messages to find an appropriate display position is not a simple task. The question is whether the message should become visible without delay when error occurs, in a status bar, or in a dialog box? As a matter of fact error management is not preferred tasks of developers and documenters. But they can't run away from this dilemma. Avoid errors

instead of managing them is the best possible approach which makes most sense and this is the simplest clarification.

Benefits

- Recovering from errors is the problem of most of the users as users cannot come into error situations.
- Errors and error messages do not suspend user's effort.
- Error messages do not confuse users
- Displaying error messages in a screen area or popup window is not necessary.

Tips:

One of the ways to find design solutions that prevent errors is to give up of "old habits" and take some rethinking. Some typical "patterns" are available to find new solutions as there are no universal rules to prevent errors. Some ideas and examples are provided below.

Some of the examples might not prevent errors themselves but help to reduce the possibilities of errors in the right directions. These ideas have been beyond on the web but they are well known to application developers.

Prevent wrong or invalid Inputs:

Numeric fields: Parsing the input string prevents users from entering letters or other invalid characters.

Data and time fields: Making available intellectual date and time fields that are preformatted or supply selection controls as an alternative of input fields (dropdown lists, spin buttons, calendar controls)

Currency fields: Utilize preformatted fields for the different units.

Prevent Invalid Actions

Disable pushbuttons that cannot be used in the current context

Do not offer functionality that is not needed (reduces complication)

Prevent Disastrous Actions

Adding explanatory texts to the respective buttons can help users to avoid severe consequences for that particular actions and informing the users about the consequences will be a good help.

Displaying confirmation dialogs will help if users might lose data

Follow the usual flow of control

Flow of control on a screen is usually from left to right and top to bottom. Users might be confused if this direction is changed indiscriminately and for example might overlook

the consequences of their actions and might get confused not knowingly how and where to progress.

Do not obscure the Screen and its Purpose

Hiding the essential information and revealing immaterial information often dominates the screen. In other cases users simply have no clue on a screen's function. Accordingly, providing the essential information and relevantly arranging the things that are predictable first helps users to know what to do on a screen and how to do it.

2.1.6. Recognition rather than recall

Making objects, events and options visible will help to play down the user's memory. It is not required that user need to memorize information from one part of the dialogue to another. Every time a suitable instruction for the use of the system is needed, they must be visible or easily retrievable [4].

An easily accessible documentation should be added as the users often cannot memorize what each object/ action/ options means so that users don't have to memorize what each object/ action/ option does[8].

Predictable results should be provided to the user actions. Designer has the responsibility to recognize the users' tasks, goals, and mental model in order to meet those prospects. Usage of the terms and images matching users' task experience so that it will be a help for the users understanding the objects and their roles and associations in accomplishing tasks.

The task of making users convinced in exploring, perceptive they can try an action, view the result, and undo the action if the result is undesirable is needed. Interfaces should be made more comfortable for the users so that their actions do not cause any irreversible consequences.

User might be interested to explore further than navigation. We need to guide them in some of the consequences for potentially hazardous proceedings. Guidelines must be extended for the proceedings performed by accident as they don't know the consequences [5].

For any inevitable motive, the vagaries of Internet communication, or as a result of error on their part, we need to guarantee that user never lose their effort. Warn the user and ask for affirmation if an action is extremely hazardous, and there is no way to undo the outcome. Such actions must be performed in extreme cases because if users receive confirmation messages commonly they might start ignoring them making them worse than ineffective [5].

User may not expect the side effect of bundling of events so it is better to avoid it. For example, only send request should be cancelled when a user choose to cancel a request to send a note. Avoid bundling an additional event, such as deletion of the note, with the cancel request. Allowing users to make their choices and their actions autonomous is the best way rather than implementing complex proceedings [5].

2.1.7. Flexibility and efficiency of use

Accelerators -- hidden to the beginner user- help to speed up the communication for the user as it can be provide flexibility to both experienced and inexperienced users.
Everyday actions will be allowed to be monitored [4].

Flexibility:
Flexibility can be defined as a way to make changes to the system so that the application is easy to use. Below are the factors that have to be taken into consideration to increase the flexibility of the system:

Task migratability: Migration means transfer between two places , in the same sense , task migratability refers to transferring the control for execution between the system and the user , meaning it is how the system supports the user to execute different tasks and vice versa .

Substitutivity: It is defined as the Input or the Output that the user provides or requires from the system.
Dialogue initiative: It is when the system allows the user to interact with it by not placing any restrictions by the input dialogue, such a situation is said to be dialogue initiative.

Efficiency:
Efficiency relates to how fast users complete their tasks once they learn how to use a application, meaning using an application more number of times increases the level of comfort and expertise in that application which helps to reduce the time taken by the user to complete a task. This can be gained by designing the application in a logical constructive manner.

2.1.8. Aesthetic and minimalist design

Information that is irrelevant or not regularly needed shouldn't be included in a dialog. When irrelevant information creeps into a dialog it in fact competes with the relevant information which diminishes the actual meaning of the dialog [4].

The importance of a program is that it might not be a piece of art but it is mandatory not to look ugly. A basic principle of what William Rotsler states is "*Never do anything of what that looks to someone else like a mistake*" [6].

A relevant example would be of arrangement of buttons on the screen. Imagine you have 4 buttons each labeled with 4 random names that have almost equal size. These buttons are displayed on the screen using an automated layout algorithm. However it is know that all these buttons are of different sizes and gives a gut feeling that the algorithm is incorporated carelessly. To eliminate this incompleteness the packaging algorithm must better understand that it is better to use the same size for all buttons that almost the same size. The principles of graphical design ought to make sense and give value to the layout algorithm now. This also applies to widget layouts done manually [6].

Another factor to pleasing the user would be to create programs that look to work faster than slackly programs carrying a sack. Only smart tricks can make a program look fast in spite the fact that you never compromised with its functionality. For example you can use for the off screen bitmap images for rendering which can then appear on a single click. Technically this is called BitBlt (Bit-Block Transfer), in the example above, the buttons can a similar concept to flicker when button is activated or screen sizes are changed [6].

2.1.9. Help users recognize, diagnose, and recover from errors

Error messages are to be user understandable and without code representation. It would suggest a solution to the end-user [4].

When an erroneous operation is performed by the user, or a regular operation creates an error (the user never knows) the program should clearly identify the error and direct the user for the next steps. Cryptic error messages and uncaught errors are an example of unhelpful error messages. Cryptic errors are those error messages that take the user nowhere. They are very less informative and will not refer to a solution to fix the error user level [9].

Uncaught errors are those errors that a program fails to recognize and moves forward with the process. But at a later point of time this uncaught error might lead to a program mal function that is not actually because of the current operation. For example a memory over usage created by one user on the server which is not caught correctly might show up its impact when another user actually tries to access the server with a different operation but by then the memory is totally engulfed and the server stops responding. Because the error is uncaught it remains a hard to trace error [9].

2.1.10. Help and documentation

To get started with any application users needs some kind basic training and knowledge. First and foremost thing is the user manual help and project documentation form these documents user can learn to access and use the application. It is vital provide help and documentation along with project. It should interface with project in such way that user can search easily [4].

User no need memorize the things which are already known to the system, such as name of the file and interface details. The system must provide the information whatever form it is [5].

System should allow for users to have a two-way interaction to illuminate or authenticate requests, or to remedy a problem. The interactive window has to be well presented and comprise good interaction features similar to other segments of interface console. For allowing users to make choice for a specific task tool should present pertinent information, provide access to related information [5].

At times the system fails to follow the user's request in spite of having objects and actions provided when they want to complete a specific task. In such cases tool can provide a two-way communication to help users achieve their target [5].

User tasks can be of a varied kind ranging from beginner to expert level. In addition to providing assistance when requested, the system should recognize and anticipate the user's goals, and offer assistance to make the task easier. To achieve the target a user wants to get, tool/system's help should be able to assist with ease and less time. Intelligent assistance should build the user independent to use to tool as the user prefers to be so [5].

2.2. System re-engineering:

Ian Sommerville has described **"***Re- engineering is a process of Reorganizing and modifying existing software systems to make them more maintainable and updating the structure and values of the system´s data*" [3].

Re- engineering has two key advantages

1. Reduced risk

There is a high risk involved in the new software development. There may be development problems, staffing problems and specification problems

2. Reduced cost

The expenditure of re-engineering is often considerably less than the costs of developing new software.

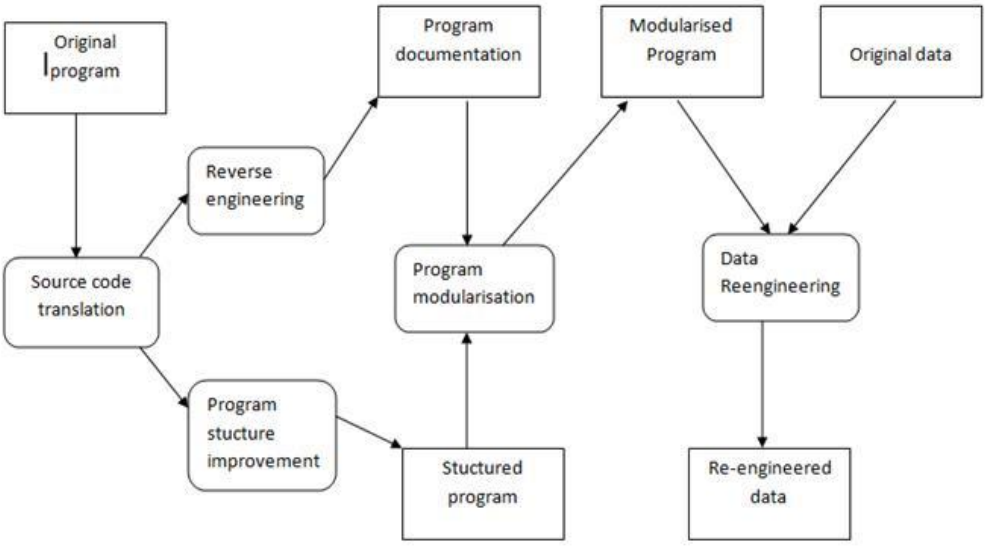

Figure: The re-engineering process

Fig2.1. the system re-engineering process [3]

The above figure illustrates the re-engineering process. The input to the process is a legacy program and the output is a structured and modified version of the same program. In this process the data for the system may also re-engineered [3].

The activities in this re-engineering process are

1. Source code translation:
In this process the program is transformed from one programming language to more advance version of the same language or a different programming language [3].

2. Reverse engineering:
In the reverse engineering stage, the complete analysis of software is done with a noted view to understand its design and specification. It may be part of a re-engineering process but may also be used to re-specify a system for reimplementation. This helps to document its organization and functionality [3].

3. Program structure improvement:
In this process the control formation of the program is analyzed and customized to make it easier to read and recognize. The program may be automatically restructured to remove unconditional branches. Conditions may be simplified to make them more readable and easy to understand [3].

4. Program modularization:

In this process basically all related parts of the program collected together and, where appropriate and redundancy is removed. It is a manual process that is carried out by program inspection and Re-organization [3].

5. Data re-engineering:
The data processed by the program is changed to reflect program changes. It involves analyzing and reorganizing the data structures (and sometimes the data values) in a program

System re-engineering may not necessarily require all of the steps which we discussed earlier Source code translation may not be needed if the programming language used to develop the system is still supported by the compiler supplier. If the re-engineering relies completely on automated tools, the recovering documentation through reverse engineering may be unnecessary [3].
Data re-engineering in only required if the data structure in the program change during system Re-engineering .anyway software re-engineering always involves some program re-structuring [3].

Re -engineering cost factors:

1. The quality of the software to be re-engineered: The lower the quality of the software and its associated documentation. The higher the Re-engineering costs.
2. The tool support available for re-engineering: it is not normally cost-effective to re-engineer a software system unless you can use CASE tools to automate most of the program changes.
3. The extent of data conversion required: if re-engineering requires large volumes of data to be converted. The process cost increases significantly
4. The availability of expert staff : if the staff responsible for maintaining the system cannot be involved in the re-engineering process, the costs will increase because system re-engineering will have to spend a great deal of time understanding the system[3].

3 Methods

The main aim of this thesis is to get clear understanding and knowledge about existing artifact manger and also get historical overview of idea. And finding out the past development and research that was done in this field.

Keeping in view the purpose of thesis, Imperative qualitative approach was used for carrying out this thesis work. As we already know our topic of interest and have preliminary knowledge about the topic and keeping in view the limitation of our process we choose the above design for our thesis work.

The study was started by formulating usability heuristics and system re-engineering concepts. By keeping the view of theoretical framework the modification level of existing artifact manger was designed. Literature on usability heuristics [4] and system re-engineering [3] (discussed n detailed sections 2.1 and 2.2 respectively) was reviewed to support and fulfill the main purpose of the thesis. Then qualitative data was collected and useful modification was proposed based on the above literature review. Results and analysis was carried out to verify and motivate the proposed outcome of the thesis work.

Our intension was to know the functionality and flow of the past artifact manger evolved and showed. And based on our literature review and current state of artifact manger and the central suggestions were determined for future directions.

This project is not restricted but our focus is to find the value added modification to existing artifact manger listed below.

- How the existing artifact manger does not fulfill the rules and regulations of the usability heuristics?
- How system re-engineering concept can be accomplished to improve the functionality of the artifact manger?

At the end of this thesis work we managed to address and find solution above two questions with implementation approach to the artifact manger. In the results we discussed about achieved projected modification based usability and re-engineering concepts and compared based on the user feedback survey technique.

3.1 Implementation

Our implementation was influenced by different factors. One of them was the modifying functionality of the existing plug-in based on usability and re-engineering concept it was a new experience to me. We had thus to explore Protégé framework and find which were the possibilities and limitations. furthermore, even though we had quite well defined

basic requirements for the application and the project, the area was still quite new and we needed to get acquainted with the domain and understand more clearly about the developing environment required to satisfy the thesis requirements for the software. Therefore we used an evolutionary software development process by trying and validating progressively each step that we understood more.

From the next section, we start to explain how we implemented the different concepts involved for improving the usability of Protégé plug-in for artifact manager based Usability Heuristics and the environment which we used to develop the functionality of artifact manger.

Our environment:

Our task required that we used an existing ontology editor like protégé, since we had experience with protégé and the enterprise otology being continued in protégé and older version of the software is also developed based on this protégé. This encourages us to choose this protégé.

Protégé is an open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontology's. We choose to implement our application as protégé with a tab-widget-plug in called artifact manger. Initially we decided to use net beans as a tool to develop and modify the source code. Unfortunately the modification does not reflect explicitly to the protégé.

The major task is to find out a suitable way of interaction with protégé Due to the lack of information, we searched through discussion groups [10] have been used finally, and then we used my eclipse 7.0 tool as a source code development environment. The developed or modified source code has been converted jar file and imported to plug-in folder of protégé to view the output.

Some best features of my eclipse 7.0

- Advanced java script tooling.
- New plug-in dash board.
- New JSF views and enhancements.

Artifact manager:

Artifact manager is software used to manage artifacts. The broader aspect of artifact manager is to create, remove, change the artifacts and search artifacts.

Functionality of artifact manager:

Artifact manager is a tab plug-in for protégé; it contains four sub tabs discussed below

The starting tab:

The "Starting" tab shown in figure 4.1 Permits the selection of the EO, it will refer the metadata for each artifact. It uses protégé standard-format file containing the ontology. The EO is integrated into the current ontology which holds artifact types and instances and other temporary classes used for the execution of the Artifact manger. If successfully imported, a message gives notice of it; otherwise an error message is given. The status of the imported ontology will be saved into the project ontology and no further importation will be permitted. Therefore the EO must not be changed. This version of artifact manger doesn't support EO change.

The types tab:

The "type" tab shown in the figure 4.2 allows adding, removing of artifacts types, on selection of an artifact type, the corresponding list of attribute associated that particular type of artifact will displayed, together with the appropriate cardinality of the attribute. We can delete a selected attribute from the list by using remove option. Similarly we can change the name of existing attribute using rename option. We can create two types of attributes like simple and complex.

The artifact tab:

This "artifact" tab shown in figure 4.3 is intended to manage artifacts. In protégé it corresponds to instances of the class of the artifact type to which the artifacts is based on. Possible actions are to add and remove an artifact. When an artifact is selected, user can give values to attributes, by creating new instances of them, finally the metadata is set by selecting (highlighting) elements from the EO on the right windowpane. The URL consists of a string that stores the physical representation of the artifact.
While adding an artifact, user can select among the different existing artifact types using dropdown list, after adding the artifact list of attribute types is displayed. Attribute types that depends of the preexistence of other attributes (when they are not directly attached to the artifact, but to another attribute for instances) will not be displayed at first, it will be necessary to create at least one instance) will not be displayed at first, it will be necessary to create at least one attribute of the triggering attribute.

Selecting an attribute in the list will display the instances of that attribute in the adjacent table. For the selected artifact from the list on the left of the screen, will appear the EO on the right windowpane. User can add whole path to metadata selection panel. User can change the path when he wants to add new one. When the complex attribute type is selected, we can create an instance of it, the value won't give effect because a complex attribute doesn't hold a value, but it will then enable the creation of contained attributes type which can hold a value.

Search tab:

The "search" tab shown in figure 4.4 allows the searching of existing artifacts, according to different possible parameters by:

- Attribute
- Ontology matching
- Both

Both matching can be combined; actually the attribute matching limits the number of artifacts that will be matched against their metadata. It should therefore perform faster than with only ontology matching.

Ontology matching can be done in two ways.

- **Part of EO selection query**
- **Free ontology query**

We first click to display the existing attribute types. Then we can select which attribute will be in the search, by selecting one of the pre-selected attribute. We can then visualize the instances of such attribute; by clicking on it we will get the list of matching artifacts.

By choosing the ontology matching way, user can choose between a free editing ontology or can extract from EO. In the latter case, one highlights which class or instance should be looked for. Then we start search, we get eventually artifacts that match. They would contain some or all of the elements from the ontology. A score gives an indication of how accurate are the search results, the user first selects in which mode the search will be performed, by free ontology or by EO selection, the additional filtration done based on threshold technique. We can see the count of matching artifacts on the top of the list box.

3.2. Identifying improvements regarding user interface of the existing plug-in:

In this section we analyzed the existing functionality and flow of the artifact manger based on the usability heuristics. In the below sections we tried to relate the implication of specific usability heuristics that use for proposed modifications to existing artifact manager tool. For each tab the applicable usability heuristics is discussed in detail in section 3.2.1 onwards.

Starting tab:

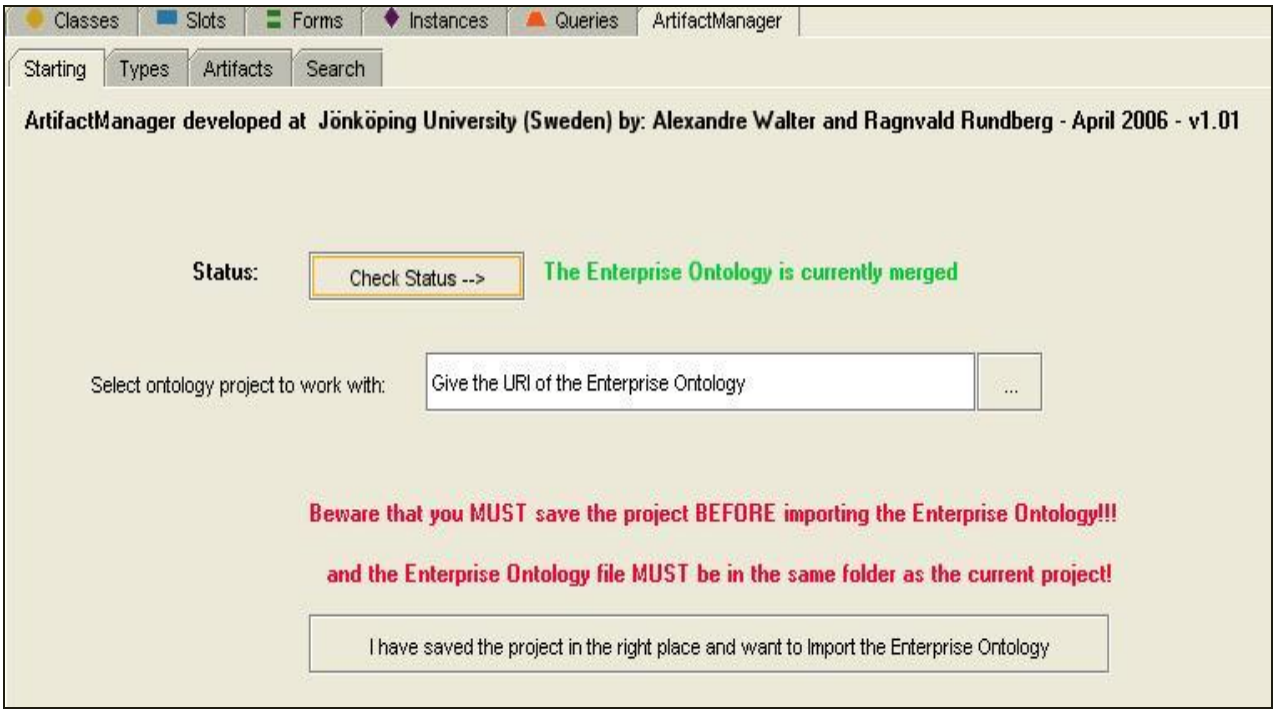

Fig 3.1. Starting tab of older version Artifact manager

From the pictorial diagram the visible minor bugs which may create an illusion or confusion to the end user are described below.

Based on usability:

1. Check status button: user need not to check the status of the ontology merging, where as it would be rational to show the status automatically loading event of the page. This modification has been proposed based on usability heuristic "**Keep the User Informed**" discussed in section 2.1.1 broadly.

2. …. Button : to give a better clarity to the end user, it is required to use proper key word as a label on the button. This modification reference has been taken from usability heuristic "**Match between system and real world"** discussed in section 2.1.2 clearly.

3. Generally across the globe **Red** is a color indicates warning or panic, in this case we assume the above informatics note should not be in red color. After importing the ontology, the "browse" button remains active for user to import ontology. This proposed modification influenced based on the usability heuristic "**Error prevention**" which we have described clearly in section 2.1.5.

Types tab:

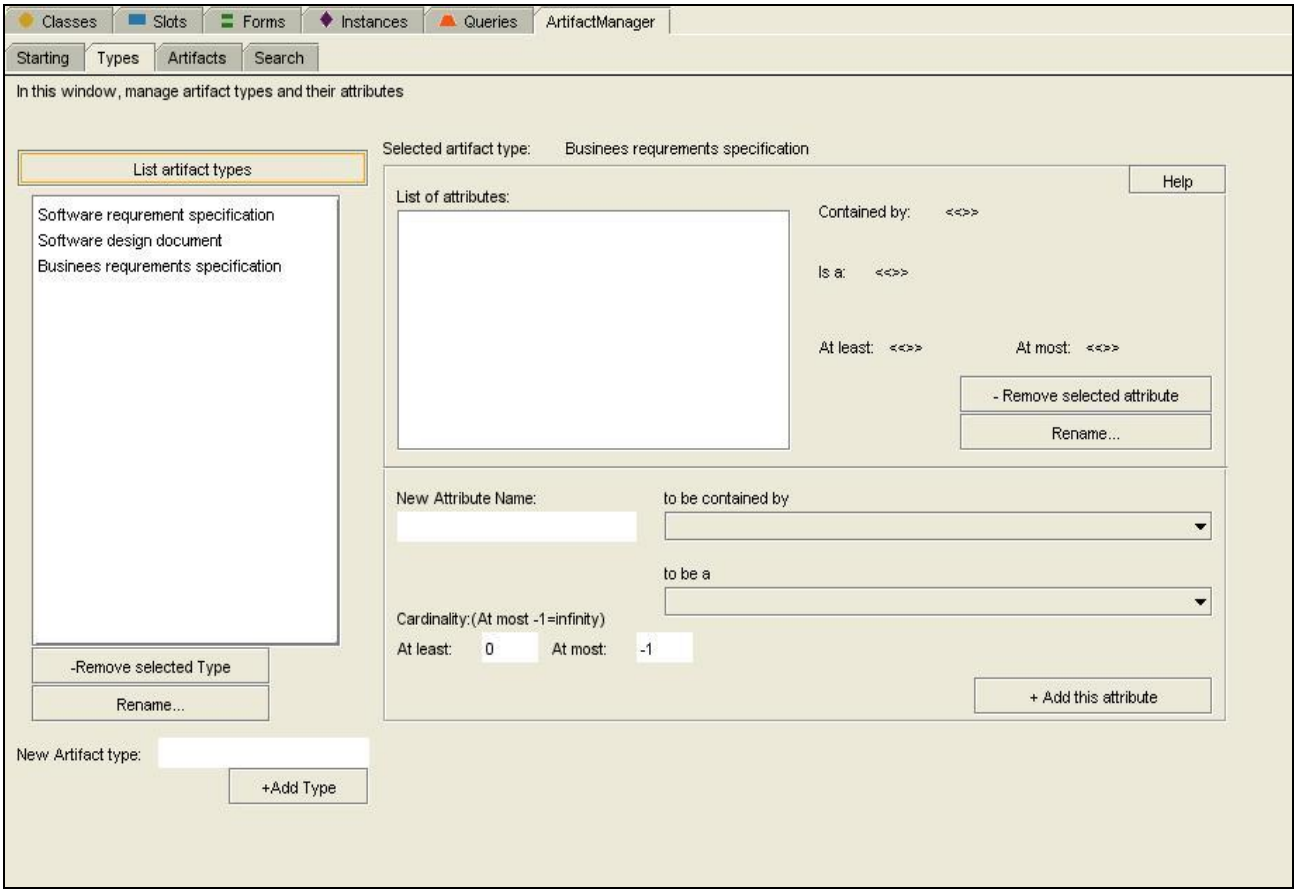

Fig 3.2. Types tab of older version Artifact manager

From the pictorial diagram the visible minor bugs which may create an illusion or confusion to the end user are described below

Based on usability:

1. List artifact types button: user need not to click the button to display the existing artifact types, this would be create a confusion if there is no artifact types entered and try to keep pressing the button to see the content. This modification has been proposed based on usability heuristics "**Keep the User Informed**" and "**Consistency and standards**"discussed in sections 2.1.1 and 2.1.4 respectively.

2. **Remove selected type**: there is no warning message displayed to the end user when he tries to delete artifact type. While deleting the artifact type it doesn't check the proper hierarchy data of artifact under available particular artifact type. This proposed modification has been done based usability heuristics **"Help users recognize, diagnose, and recover from errors**" and "**error prevention**" discussed in section 2.1.9 and 2.1.5 respectively.

3. Showing all unnecessary information to the end user create ambiguity. This modification proposed based the usability heuristic "**Recognition rather than recal**l" described in section 2.1.6.

Artifacts tab:

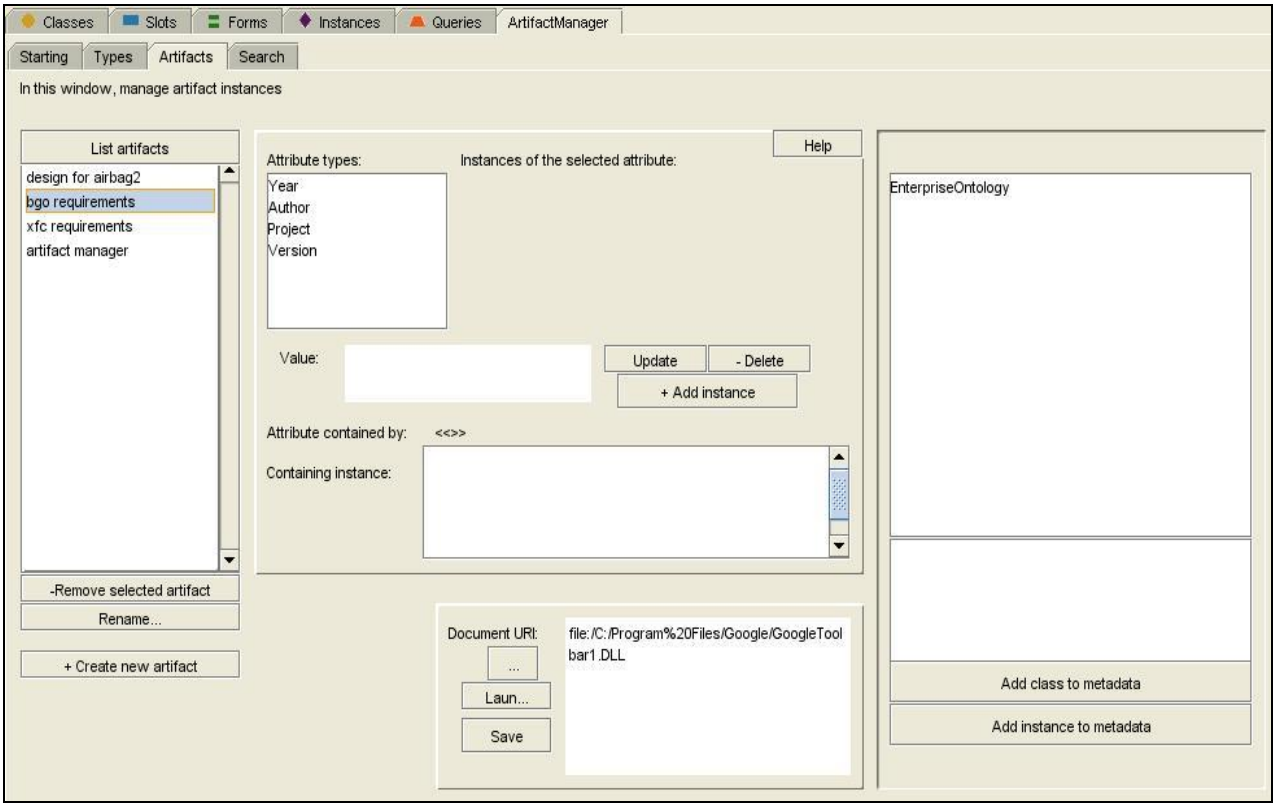

Fig 3.3. Artifacts tab of older version Artifact manager

From the pictorial diagram the visible minor bugs which may create an illusion or confusion to the end user are described below

Based on usability:

1. List artifacts button: user need not to click the button to display the existing artifacts, this would be create confusion if there is no artifacts. This modification has been proposed based on usability heuristics "**Keep the User Informed**" and "**Consistency and standards** "discussed in sections 2.1.1 and 2.1.4 respectively.

2. Remove selected artifact: there is no warning message displayed to the end user when he tries to delete existing artifact. While deleting the artifact there is no precautionary message is displayed. This proposed modification has been done based usability heuristics **"Help users recognize, diagnose, and recover from errors**" discussed in section 2.1.9 clearly.

3. Showing all unnecessary information (buttons, labels, text box, and list box) to the end user create ambiguity. This modification proposed based the usability heuristic "R**ecognition rather than recall**" described in section 2.1.6.

4. **"…."** And "**Laun"** Buttons: to give a better clarity to the end user, it is required to use proper keyword as a label on the button. This modification reference has been taken from usability heuristic "**Match between system and real world"** discussed in section 2.1.2 clearly.

Based on Functionality:

1. While trying to add the classes to metadata. It is time consuming task to add each and every class to metadata as a separate event. There may be a chance of missing some important class to be added .it may end up with fragment of system with inaccurate results.

Search tab:

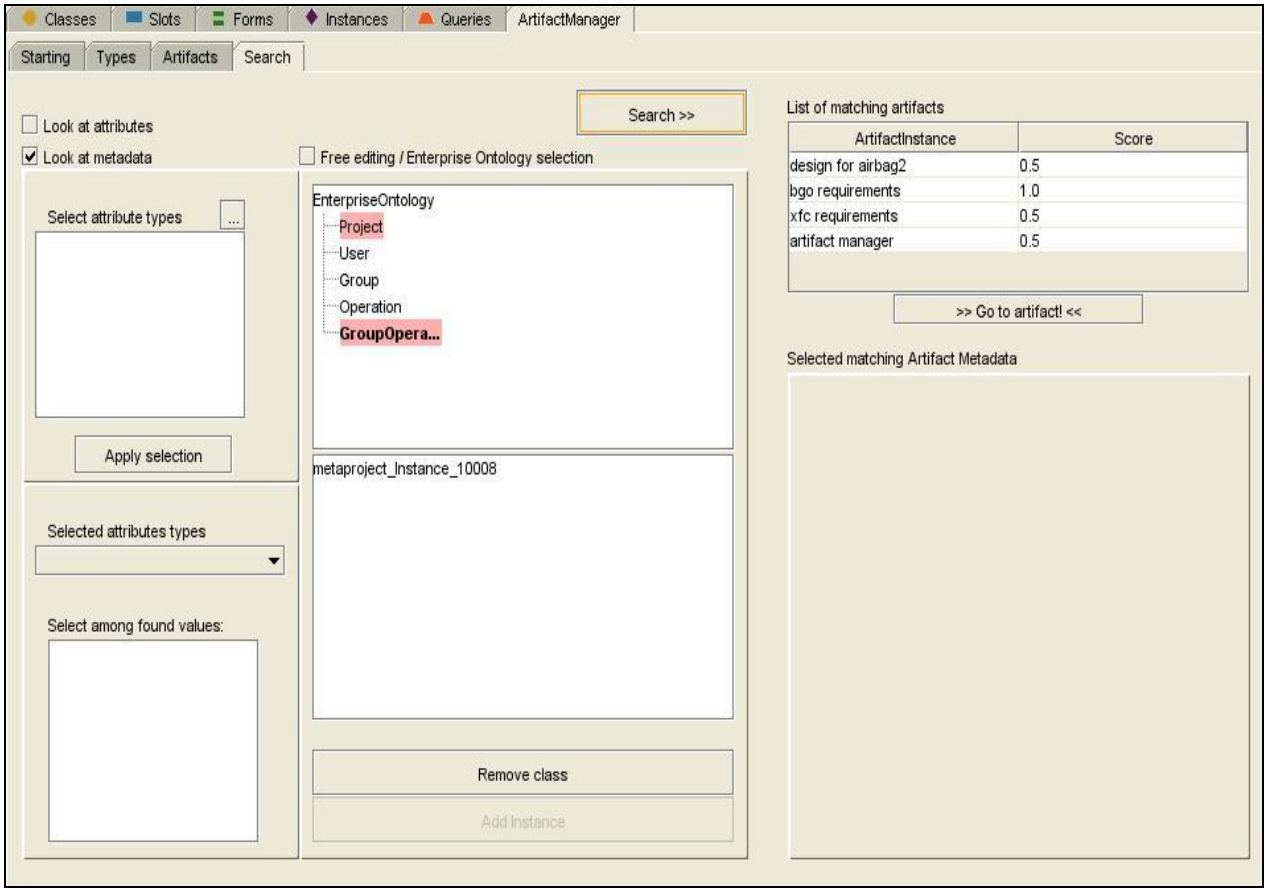

Fig 3.4. Search tab of older version Artifact manager

From the pictorial diagram the visible minor bugs which may create an illusion or confusion to the end user are described below

Based on usability:

1. … Button: user need not to click this button to display the exciting select attribute types, this would create confusion to end user. . This modification has been proposed based on usability heuristics "**Keep the User Informed**" and "**Consistency and standards** "discussed in sections 2.1.1 and 2.1.4 respectively.

Based on Functionality:

1. While trying to add the classes to enterprise ontology selection. It is time consuming task to add each and every class as a separate event. There may be a chance of missing some important class to be added .it may end up with fragment of system with inaccurate results
2. It is all ways better to know the list of matching artifacts count, it will use full to filter the data and easy to find useful matching artifacts.
3. It is helpful to extract or sort the data based on score provided by the user.

4. Results

This section consists of three segments. In the first segment we have described achieved usability improvement of existing artifact manager tool based on usability heuristics. In the second segment we described the improvement of existing artifact manager tool based on system re-engineering theory. Finally we have discussed the overall result achieved by applying usability heuristics and system re-engineering.

4.1 Improved Usability

In this section we have described visibly about improved usability by applying usability heuristics on existing artifact manager tool.

Starting tab:

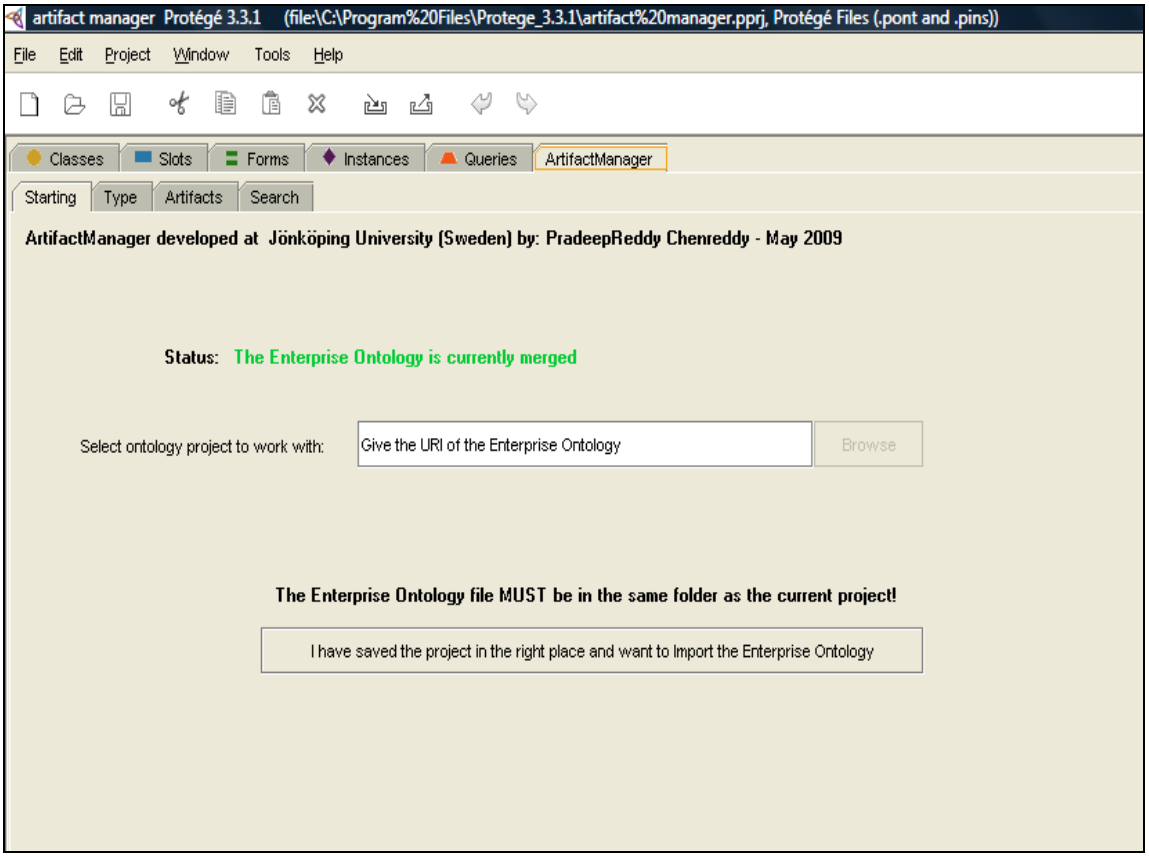

Fig 4.1. Starting tab of modified version artifact manager

As earlier we have discuss the excising bugs and functionality based on that we have proposed solutions to existing bugs some shown in above picture disused below.

Accomplished results:

1. The Check status button has been removed, the functionality of the status display has-been modified such that the status will automatically display while loading event of the page.
2. Proper label name has assigned to …. Button.
3. The text message color has been changed to visual friendly color.
4. Static information's is being removed from the screen and validation is checked and alert message has been displayed on click event of browse button.
5. After merging the ontology the browse button has been disabled to prevent end user to select new ontology.

Types tab:

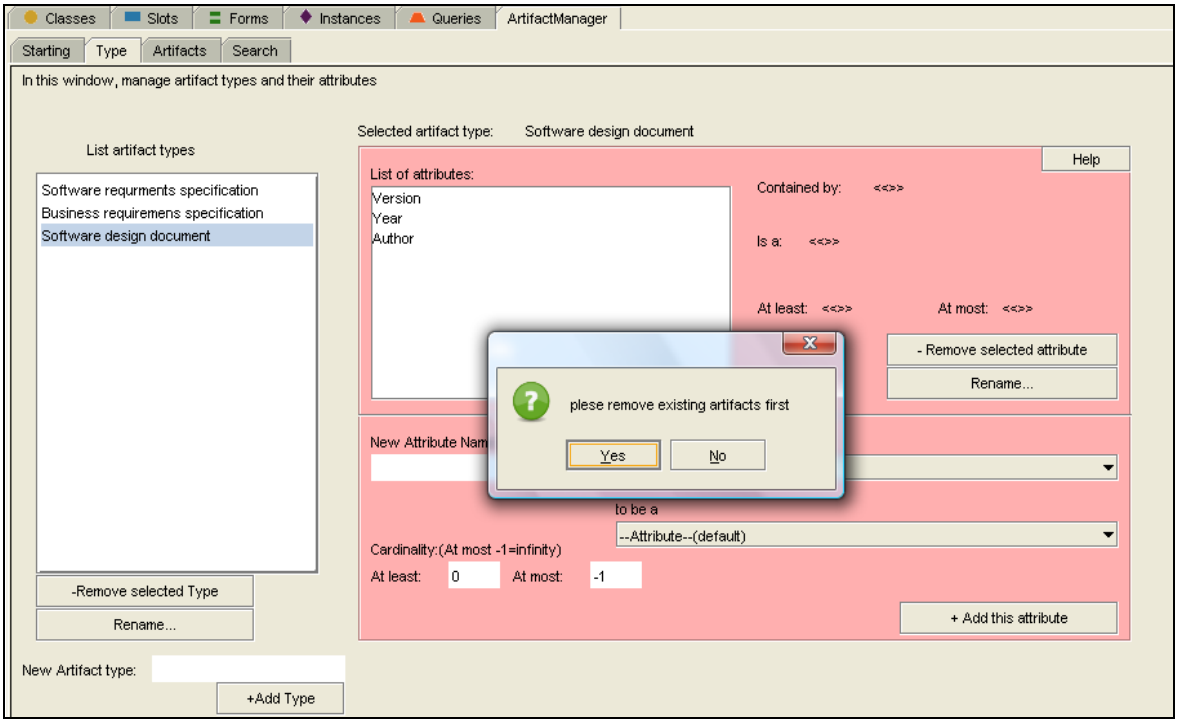

Fig 4.2. Types tab of modified version artifact manager

As earlier we have discuss the excising bugs and functionality based on that we have proposed solutions to existing bugs some shown in above picture disused below.

Accomplished results:

1. List artifact type's button has been removed. The functionality of the list artifact type's button has-been modified such that the status will automatically display while loading event of the page.

2. On click of Remove selected button a popup warning message displayed to the end user to get the conformation for removal. On back ground condition has been checked for the artifact type whether it contains any artifact or not. In case it contains any artifacts then user will not able to delete artifact type

3. Proper hide and when is given to the unnecessary information to the end user to reduce ambiguity. (considering one example on selection artifact type, the corresponding information show to the user)

Artifacts tab:

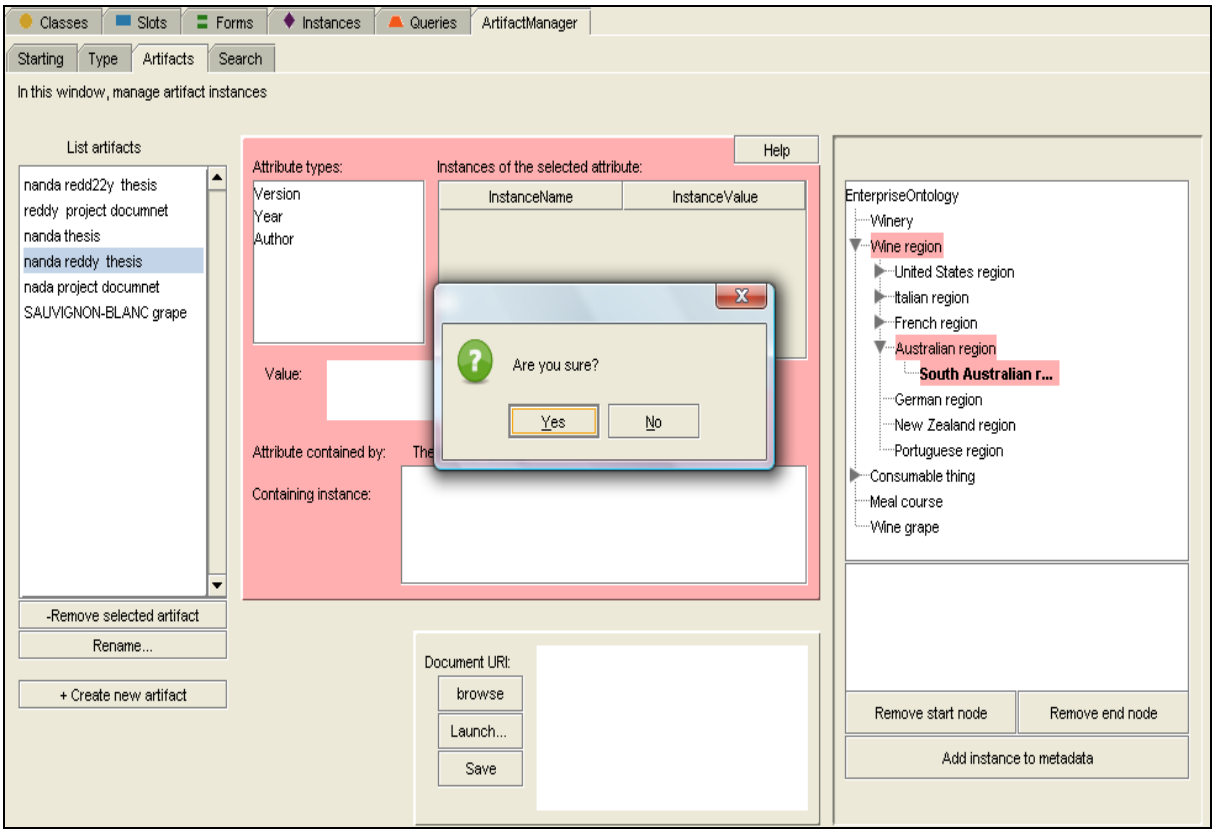

Fig 4.3. Artifacts tab of modified version artifact manager

As earlier we have discuss the excising bugs and functionality based on that we have proposed solutions to existing bugs some shown in above picture disused below.

Accomplished results:

1. List artifacts button has been removed. The functionality of the list artifacts button has-been modified such that the status will automatically display while loading event of the page.
2. On click of Remove selected button a popup warning message displayed to the end user to get the conformation for removal.

3. Proper hide and when is given to the unnecessary information to the end user to reduce ambiguity. (considering one example on selection artifact, the corresponding information show to the user)

4. Now we are able to add the whole path to metadata. Its help full to time reduction. Possibility of missing important classes has been reduced. It will useful to achieve accurate results.

5. **"…."** And "**Laun"** Buttons label name has been changed to a meaning full name.

Search tab:

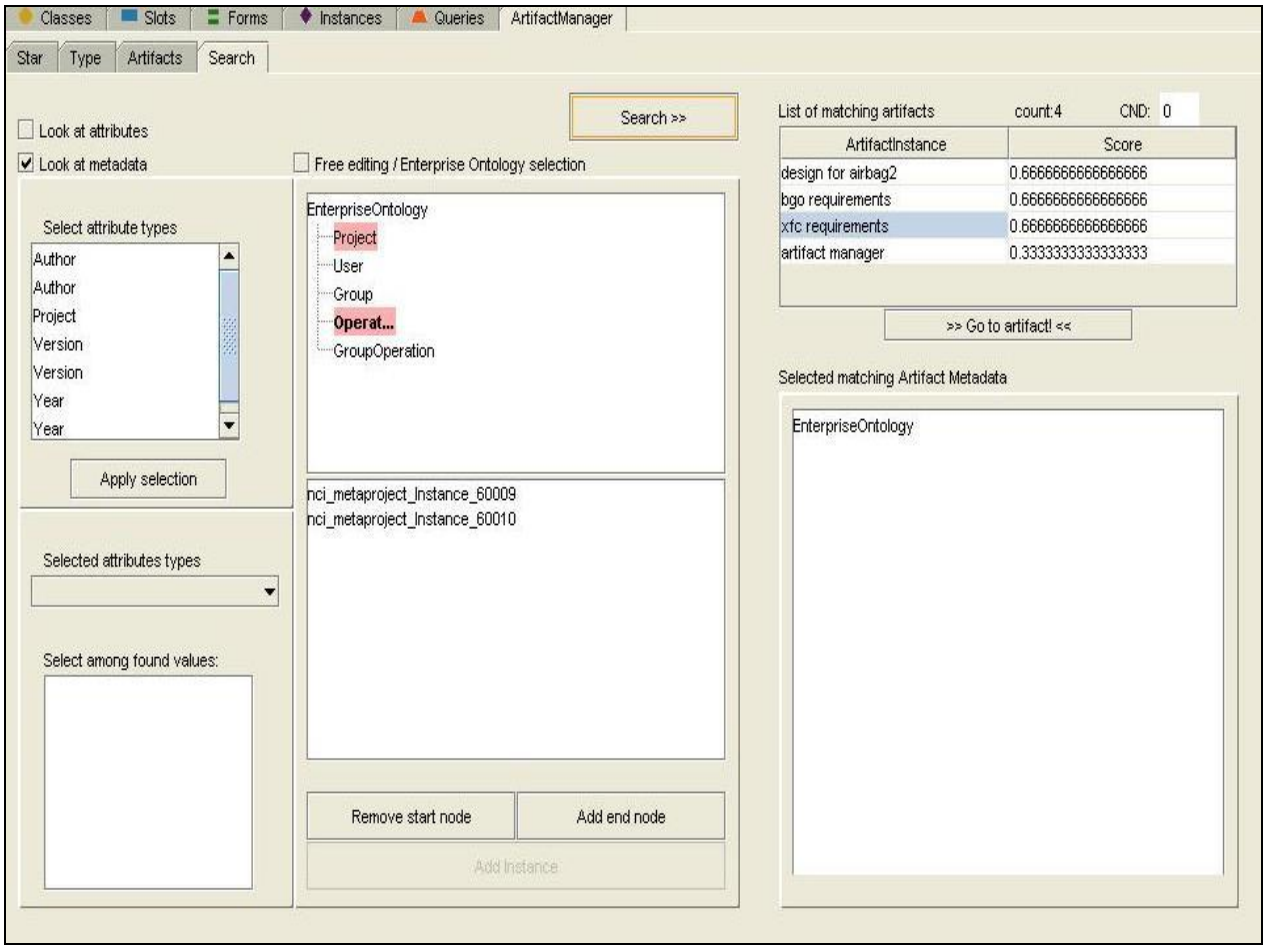

Fig 4.4. Search tab of modified version artifact manager

As earlier we have discuss the excising bugs and functionality based on that we have proposed solutions to existing bugs some shown in above picture disused below.

Accomplished results:

1. … Button has been removed. The functionality of the … button has-been modified such that the status will automatically display while loading event of the page
2. Now we are able to add the whole path to enterprise ontology selection its help full to time reduction. Possibility of missing important classes has been reduced. It will useful to achieve accurate results
3. Additional feature has been incorporated to show the count of the list of matching artifacts .certainly it will help the user to see the number of matching artifacts
4. Finally user can filter and manipulate the matching artifacts with the help threshold functionality .(user require to fill threshold functionality column with required score to display the specific matching artifact)

4.2. Results based on reengineering:

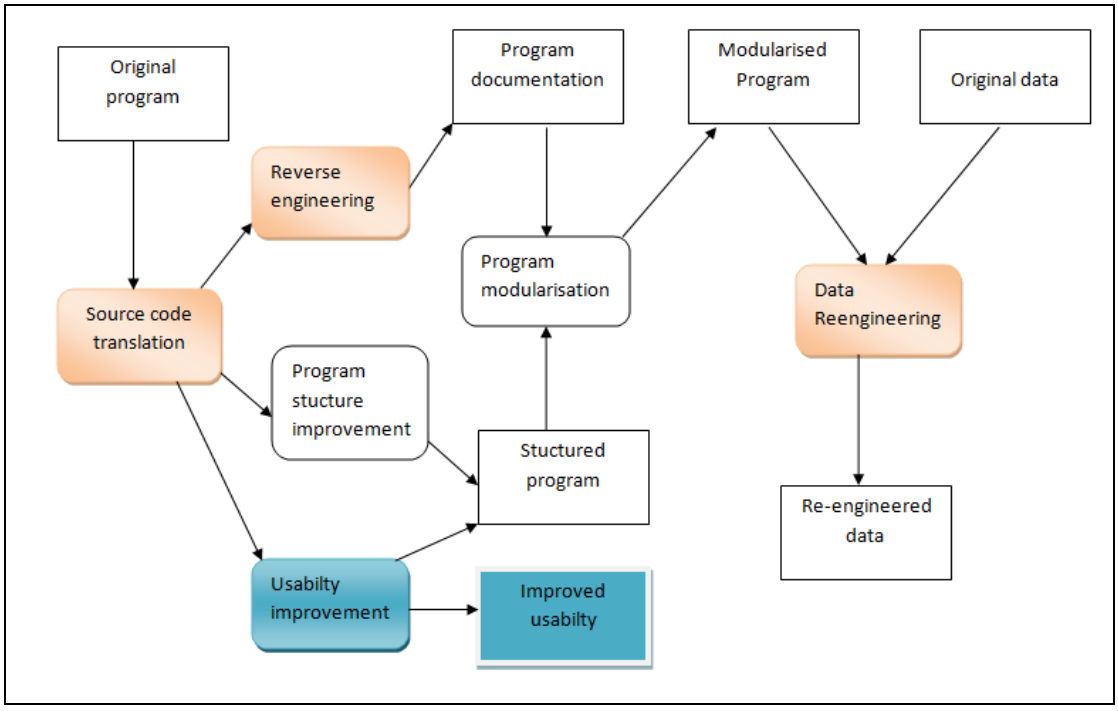

Fig 4.5. Applied re-engineering process diagram

As illustrated above diagram the notation of software reengineering has five phases. Not necessarily all the five phases required to be accommodated in any particular software reengineering process.
Here we have used three major phases of reengineering marked with colored box, and explanation of usage of these particular phases is described below.

Source code translation:

Actually In this phase program should modified from one language to another programming language or old version to advanced version of that same programming language. In this occasion we have modified few sections of old programming language to advance version of that programming language. The major development and design component used in old version of software was core java. We all know core java has its own limitation to explore all functional and requirements. The current scenario compelled the developer to look forward advance version of java technology such as j2ee and JSP. User interface which were designed in java applet has been modified and replaced by advanced java concepts such as java swings.

While working in this phase we had some complications, especially to figure out suitable environment for developing software. In the document of old version software doesn't provide the clear information of the environment .for this reason we spent lot of time to set up an environment. We could not manage to find the appropriate environment for the software development, but finally we have chosen my eclipse 7.0 version to develop and to do modification of that software.

We have done few modifications in the source code in the various stages to achieve the required functionality; below we have given the example code which shows achieved modifications.

Source code of old version artifact manger:

```
Public void saveClassMetaData()
                {
                ProtegeClass pcls = getSelectedClass ();
                                If (null! = pcls)
                                {
                                                If (! isHighLighted (pcls))
                                                {
                                                                m_collClassMD.add (pcls);
                MetaDataManager.setClassMetaData (m_ains, m_collClassMD);
                                                }
                                                Else
                                                {
                                                m_collClassMD.remove (pcls);
                MetaDataManager.setClassMetaData (m_ains, m_collClassMD);
                                                }
        }

                                                RefreshClassTree ();
                                UpdateClassButton ();
    }
```

Source code of new version artifact manger:

```
Public void saveClassMetaData(ProtegeClass ParentSelectedClass)
                {
                ProtegeClass pcls = getSelectedClass();
                                if( null != pcls )
                                {
                                if( !isHighLighted( pcls ))
                                {
                                m_collClassMD.add(pcls);
                                for(int i=0;i<=10;i++){
                                pcls=pcls.getFirstParent();
                                m_collClassMD.add (pcls);
                If(ParentSelectedClass.compare(pcls)       pcls.getTopParent().compare(pcls))
            {
                                break;
                                }
                                 }
                                childNode=true;
                MetaDataManager.setClassMetaData( m_ains, m_collClassMD );      }
                else
                {
                m_collClassMD.remove( pcls );
                for(int i=0;i<=10;i++){
                pcls=pcls.getFirstParent ();
                m_collClassMD.remove (pcls);
                If (ParentSelectedClass.compare (pcls) || pcls.getTopParent ().compare (pcls)){
                                Break ;}
                                ChildNode=false;
MetaDataManager.setClassMetaData (m_ains, m_collClassMD );
                                                }
                                 }
                                refreshClassTree ();
                                UpdatablessButton ();
        }
```

In the old version of artifact manager while adding path to metadata we have to add each and every single class and remove a single class. It is time consuming task to add each and every class to metadata as a separate event. There may be a chance of missing some important class to be added .it may end up with fragment of system with inaccurate results. To solve this problem we have created on option which first checks the parent class, if the user selects the child class then whole path added to metadata.

Reverse engineering:

In this phase of software reengineering we had done a thorough study on the working functionality of old software, during this comprehensive study we found various functionality which can be improved further to simplify the user access and reduce the complexity. This additional functionality will be value-added to this existing plug-in.
Based on the findings during analysis and study phase of the software, we have modified user interface design and code to an extent such that it would incorporate the suggested

changes and certainly helpful to the end user. This user friendly approach done based on Usability Heuristics .Certain principles of Usability Heuristics. Such as displaying proper error message to the end user at various stages to minimize the mistakes and enabling end user to have more command and control on the software.
Usability Heuristics improvise the performance and flexibility of the software so that end user can learn and handle precisely

The modified user interface design is shown in section 4.1 (fig numbers 4.1, 4.2, 4.3 and 4.4) if you draw a comparison between old user interface design shown in section 3.1 (refers fig numbers 3.1, 3.2, 3.3 and 3.4) we can see a visible change in user interface design between new version and old versions of artifact manger.

For example in old version search tab (fig 3.4) there was no option to see the count of matching artifacts list, in the old version to add a path to enterprise ontology selection we have to add each and every single class. There was no option to filter the matching artifacts list. Attribute list was displayed by the click of "**…**" button

On the new version the visible modified user interface design changes are.
We have introduced one label to display the count of matching artifacts list and one input field enter the value for thresh hold technique to filter the data. We added two new buttons to enterprise ontology selection frame to add whole taxonomic path to search selection. We removed "**….**" Button but we can see the list of select attribute types on the event of page loading

Data re-engineering:

In this phase we have done two major modifications to the program to get accurate search information .Those main changes are reflected in search tab of the plug-in and it helps to get accurate search result using threshold condition and displaying the total count of artifacts. And another reflection is adding whole path to metadata panel in artifacts tab and adding whole path to search selection panel in search tab.

Usability improvement:

As shown in figure 4.5 marked with blue color the usability improvement is integral part of ongoing system re-engineering of this artifact manager. In this case the Re-engineering process starts with existing artifact manager tool to a new transformed and improved version of artifact manager. In this process various activities such a usability improvement, value added functionality changes have been achieved. As we have described in the diagram usability improvement is derived from "source code translation". The necessary source code has been modified based on the proposed changes by applying usability heuristics on existing artifact manger. Thus we can conclude that usability improvement is an integral part of the system reengineering.

Hence with this usability improvement we have improved the usability of the existing artifact manager and improved the structure of the program in certain areas where ever the code modification has been done.

4.3. Final results:

In this section we have discussed how the implementation have done based on usability heuristics and how we modified the software based on reengineering concept introduced in the theoretical background.

Protégé is a powerful tool for dealing with ontology's. It can create and manage ontology's in an effective manner. Artifact manger meant to handle metadata (which as references to ontology) therefore it appeared to be a rational choice to reuse its environment and implement a plug-in for it. Using ontology to store and manage artifacts seemed a natural and best option since it enables the creation of complex structures for easy handling and understanding.

Importing a project is made by using protégé functionalities for including and merging projects .previously they implemented directly the methods they use to make it integrated in our plug-in. we tried to modify This functionality but it was difficult to implement through but works satisfyingly if we take apart same bug we discovered in protégé or limitations like the necessity to have the EO file in the same folder as the current project. Protégé is a university project, and is still under active enhancement and development.

We have found previously developed plug-in working sufficiently but after doing vivid study on that plug-in, we came across some limitations and decided to improvise the functionality and usability by providing suitable solutions. appropriate steps has been included to achieve version compatability.In this process of improving usability of plug-in, we accomplished to see the status of enterprise ontology merging in the "starting tab "on the page loading event. In the "types tab" user has been able to these the existing artifacts types successfully on the event of page loading and user has been prevented delete artifact type in case it contains any artifacts in artifacts tab.

In the artifacts tab user has been able to see the existing artifacts on the event of page loading and proper error message has been provided to safeguard to the end user, and we succeeded to add the whole path metadata. In the search tab we manage to see the existing attribute types on the page loading event. Also we are able to add the whole path to enterprise ontology selection and Additional feature has been incorporated to show the count of the list of matching artifacts. Finally user can filter and manipulate the matching artifacts with the help threshold functionality.

In this enhancement area we have taken various references of usability heuristics to achieve above mentioned results. Few vital usability heuristics that we have

accommodated were Visibility of system status, Help and documentation, Aesthetic and minimalist design and Error prevention. Apart from usability heuristics we tried to put software reengineering concept to improve the functionality and performance of the existing plug-in.

Above discussed results were achieved by applying the system re-engineering process. Initially we have done study of the existing artifact manager and proposed the necessary transformation in improvement with help of usability heuristics. In execution phase (refer figure 4.5) each aimed result was started from source code translation phase to improved usability and data re-engineering phases. In source code translation phase we have modified the source code based on inputs proposed from the system re-engineering and usability heuristics. Source code translation, resulted the improvement in structure of the program, usability and functionality. Usability improvement further strengthens the structure of the program.

For the future work the developed plug-in should support to upcoming versions of ontology and creation of a database to store the information to increase performance of the artifact manger tool (presently we are using same folder to store the data to import ontology).as of now we have not modified searching algorithm to find the artifact matching's. We can use other matching algorithms that could be helpful to find the proper searching results.

4 Conclusion and discussion

Our work is to modify the functionality and increase usability of an existing plug-in for the protégé. We have used java and advanced java language to modify functionality and we used my eclipse tool as design development tool. Essentially the fundamental concept to improve usability we used usability heuristics and to improve the functionality we used software reengineering concept.

The modified version of artifact manger has clearly succeeded to avoid to ambiguity of end user by removing unnecessary pictorial representation of buttons. Display the proper error message when it's needed. Provided some flexible option to add the path to metadata and introduced some threshold technique to get precise search results.

To get bug free results we have followed the standard procedure of the software re-engineering process starting from source code translation, usability improvement to program structure improvement. Every aimed and achieved result has followed the few activities within the boundary of software re-engineering.

We have modified the structure of the existing plug-in in such a way that, it can enable for the further development and simplification by adding new functionalities.

5 References

[1] Andreas Billig, Kurt Sandkuhl: Enterprise Ontology based Artefact Management. GI Jahrestagung (2) 2008: 681-687, Lecture Notes on Informatics.

[2] Billig A. and Sandkuhl, K. (2002) "Match-Making based on Semantic Nets: The XML-based BaSeWeP Approach." Proceedings XSW 2002, Springer Verlag

[3] Ian Sommerville, Software engineering-7th Ed, System re-engineering *ISBN* 0-321-21026-3

[4] Ten Usability Heuristics by Jakob Nielsen
http://www.useit.com/papers/heuristic/heuristic_list.html(Acc. 2009-06-01)

[5] Dokeos User Interface Guidelines 1.0
http://www.dokeos.com/doc/duig-1.0.pdf(Acc. 2009-06-01)

[6] A Summary of Principles for User-Interface Design by Talin
http://www.sylvantech.com/~talin/projects/ui_design.html(Acc. 2009-06-01)

[7] Match between System and the Real World Definition
http://coweb.cc.gatech.edu/cs2340/5466(Acc. 2009-06-01)

[8] Recognition Rather than Recall Definition
http://coweb.cc.gatech.edu/cs2340/5468(Acc. 2009-06-01)

[9] Help Users Recognize, Diagnose, and Recover from Errors Definition
http://coweb.cc.gatech.edu/cs2340/5469(Acc. 2009-06-01)

[10] The protégé-discussion Archives
https://mailman.stanford.edu/pipermail/protege-discussion (Acc. 2009-01-08)