Chapter 7
Design and Validation of Agile Information Radiator
(AIR)
7.1 Introduction

It is often emphasised that tools designed in support of agile processes should be tailored for the specific needs of the development team (Kelter et al., 2003; Chin, 2004). However, agile supporters constantly stress that agile methods can be successfully applied exclusive of supporting tools (see Highsmith (2000, 2004) for example). While this assertion sounds plausible, there is little doubt that if properly applied, these tools may add value to the agile development environment. Several reasons exist that necessitate tool support for agile software development; larger projects are often said to induce complexity, and managing complexities is seldom achieved without support tools (De Souza et al., 2005). In larger projects, process data and team activities are often managed through project management tools (Kelter et al., 2003). In addition, the use of tools offers a means of quantifying development practices (although its value depends on the usefulness of the measures), while allowing traceability (Behrens, 2006). Furthermore, tool support may become essential for agile distributed teams (Angioni et al., 2006) as these teams can be distributed across the globe and major construct of agile manifesto of co-located teams, face to face interaction seems to be almost impossible to implement, therefore it is not possible to synchronise the development process and team members.

Tools supporting the processes recommended by agile methodologies fall into three categories (Highsmith, 2004): collaboration, technical information sharing, and project management

7.2 Information Radiator

Cockburn (2002) has stated that “Software development is a goal based game of invention and communication”. He coined the term “information radiator” in 2000, and introduced it in his book ‘Agile Software Development’ in 2001. An information radiator is something that interested parties can use to find out about the state of the project without needing to interrupt or bother the team members. An example could be a graph of stories that have been identified vs customer tests that have passed over time. Ideally these radiators should be physical reports printed and displayed in a prominent place in the project work space. Information Radiators are useful quite simply because they provide an effective way to communicate project status, issues, or metrics without
a great deal of effort from the team. The premise is that these displays make critical, changing information about a project accessible to anyone with enough ambition to walk over to the team area and take a look.

Information Radiators are also good ways to remind the team of critical items, such as issues that need to be addressed, items on which the team is currently working, key models for the system on which they are working, and the status of testing. Depending on the type of information tracked on the Information Radiators, these displays can also help the team to identify problems early.

With distributed teams it is possible to achieve similar results as in collocation with electronic distribution. While there are benefits with distributed development, it comes at a cost of increased complexity in coordination, communication, and collaboration. These increased complexities can be addressed via tool support to some extent. For such tools to be useful for agile teams, they should also provide support for specific agile practices and help overcome some potential drawbacks in agile methods. So the role of electronic information is more than it played in co-location. Agile Information Radiator (AIR) has to manage three Cs which comes with distribution i.e. coordination, communication, and collaboration because without these three Cs distributed development with agile practices is not possible.

7.3 Requirement Extraction for Agile Information Radiator (AIR)

The initial step is the selection of agile methodology which can be taken as the base for the development of this Agile Information Radiator (AIR). As it is clear from our survey that 73.5% organisations operating in India are using Scrum methodology, Scrum is the most prominent agile methodology used in India. Scrum itself does not provide any software development practice rather than it provides project management practices.

The next step was the extraction of requirement for development of the tool. For this purpose, two techniques are used, first is the outcome of the survey as this survey helps us in understanding about the communication, knowledge sharing and documentation process followed by organization which is a significant input for designing this tool. Secondly, detailed interviews and discussion sessions were conducted with
professionals of our sample organisations to understand their requirements and expectations from the tool. Professionals were asked about the development, documentation and management process. Requirements are gathered through three perspectives: Temporal, work-item and Selection of work.

7.3.1 Temporal Perspective

Temporal perspective portrays a project from time horizon dimension. From a planning perspective, temporal prospective deals with the time span which should be planned ahead of task. Temporal perspective of project and sprint share many common attributes but are serving different purposes altogether where the temporal perspective is a long-term planning, iteration is a short term planning. The project temporal prospective has a set of responsible persons, a deadline and a number of planned hours. The number of planned hours shows how many hours were actually planned for the project. Projects usually last for at least several months and can last for more than a year. The requirements in the project contain the requirements that must be implemented during the project. The tasks in the project contain the tasks that have been generated for the project.

The iteration temporal prospective is a short-term planning with a fixed deadline. In scrum these iterations are also called sprints, A project can have many sprints. Iterations are always done in the context of a project.

7.3.2 Work Item

Three different work items were identified. The first is the requirement, the second is the task and the third is the effort spent. Requirements are high level abstraction of the system to be made. It describes what is desired from the system. Most often a requirement describes some features that the software needs to have. Each requirement has a priority attached with it (cf. Table 7.1). The task is a low level description of the requirement. It describes what is needed to be done. Single task can be related to one or multiple requirements. A single requirement can also be broken into multiple tasks (cf. Table 7.2)
### Table 7.1: Properties of Requirements

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible</td>
<td>The person who is responsible for the implementation of the requirement.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority of the requirement (different organisations have their own priority system)</td>
</tr>
<tr>
<td>Links to specification documents</td>
<td>Links to documents in external systems that describe the requirement more precisely.</td>
</tr>
<tr>
<td>Originator</td>
<td>The person who is the originator or creator of the requirement.</td>
</tr>
<tr>
<td>Attached specification documents</td>
<td>Links to documents kept in the tool that describe the requirement more precisely.</td>
</tr>
</tbody>
</table>

### Table 7.2: Properties of Task

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>One responsible</td>
<td>The person who is responsible for the implementation of the task.</td>
</tr>
<tr>
<td>Many responsible persons</td>
<td>Several persons who are responsible for the implementation of the task: this property is a subset of the many responsible persons with weights property, but here each person has an equal weight.</td>
</tr>
<tr>
<td>Overdue flag</td>
<td>A task is marked overdue if the deadline of the iteration or project it belongs to has passed and the task has not been implemented.</td>
</tr>
<tr>
<td>Implementation status</td>
<td>The implementation status of the task: status can be one of new, open, reviewable or closed.</td>
</tr>
<tr>
<td>Original estimate</td>
<td>The original estimate of the effort needed for the implementation of the task.</td>
</tr>
<tr>
<td>Remaining effort</td>
<td>An estimate of the remaining effort needed for the implementation of the task.</td>
</tr>
<tr>
<td>Percentage done</td>
<td>An automatically calculated estimate of the work completion percent: calculated from the remaining effort and the original estimate: can be over 100% if the original estimate is lower than the remaining effort.</td>
</tr>
<tr>
<td>Deadline</td>
<td>The date when the task must be done.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority of the task related to the other tasks in the same container of work.</td>
</tr>
<tr>
<td>Creator</td>
<td>The person who created the task.</td>
</tr>
<tr>
<td>Reviewer</td>
<td>The person who is responsible for reviewing the task.</td>
</tr>
<tr>
<td>Change history</td>
<td>A log of all changes that have been done to the task or its properties.</td>
</tr>
</tbody>
</table>
The third piece of work is the effort spent entry (cf. Table 7.3). This does not provide any software but this helps in managing the resources. Effort spent entry tells us how much time has been allotted, how much have been spent. What is the status of the project, how much time has been left for working. The effort spent entry is always related to a task. Effort spent entries are primarily used to generate effort spent reports.

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>Number of hours of work that has been performed.</td>
</tr>
<tr>
<td>Work type</td>
<td>One of the work types listed in a companywide list of possible work types: work types can be added to and removed from the list when required. Work type can include development, testing, impediment, documentation.</td>
</tr>
<tr>
<td>Recorder/Monitor</td>
<td>The person recording the performed work.</td>
</tr>
</tbody>
</table>

7.3.3 Selection of Work

We found that usually there are two levels where selection of work is done. First or higher level deals with the selection of requirements which have to be implemented. Mostly the order and priority of implementing of requirements are chosen based on the wishes of the client.

The lower or second level of work selection is selection of tasks (user stories), which are subset of requirement selected above. Usually tasks are selected after discussing in the team meetings with leader (In scrum, scum master monitors the team). Set of tasks are groomed and selected for the iteration. Selections of tasks are based on several priority factors like currently unfinished work, most profitable tasks, client priorities etc. An individual programmer usually selects the next thing he should do from the list of tasks in the ongoing iterations. The tool must notify the person who is set as responsible for a task by e-mail.

7.3.4 Monitoring Progress

For monitoring of the progress of the project, regular meetings are held by the teams. In these meetings each individual shares the status of the task assigned to him, any difficulties he/she faced or other discussions are held in different type of meetings. Burn down chart is another way to measure the progress of the team. Burn down
chart/graph is explained in the next section. The progress of work is monitored either by iteration burn-down graph or sprint burn-down graph.

7.3.5 Roles

There are core and ancillary roles. Core roles define the responsibilities of those persons who are committed till the end of the project but ancillary roles are just involved for some particular time period in the project.

7.3.5.1 Core Roles

There are three core roles in Scrum Development:

- **Product Owner**

  The Product owner plays the role of the stakeholder and is represents the customer’s voice, means, what in actual a customer wants to develop. He is accountable for the quality and value work to the business. He writes the user stories based on their requirements then prioritize these and save in the backlog of the system.

- **Scrum Team**

  Scrum team plays vital role in the delivery of the product on time. It is responsible for delivering potentially shippable product increments at the end of each Sprint. A team can be defined as the group of those persons who do the actual work like analyze, design, develop, test, technical communication, documentation etc. Teams for development work are self organizing and cross functional by nature.

- **Scrum Master**

  Scrum master is the person who removes impediments (obstacles) and makes sure that the team is fully functional and productive. The Scrum Master is not the team leader, but acts as a shock absorber between the team and any distracting influences. The Scrum Master ensures that the Scrum process is used as intended. A key part of the scrum master is to monitor and keep it focused on the tasks at hand. He guards the team from the external interferences.
7.3.5.2 Ancillary Roles

The persons who do not have any formal role throughout the project but they participate in the project from time to time according to their responsibilities.

- **Stakeholders**

Customers, vendors or clients who are involved in the project are called Stakeholders. They are people who enable the project and for whom the project produces the agreed-upon benefits that justify its production. They are only directly involved in the process during the sprint reviews.

- **Managers**

Duty of the managers is to control the environment so that quality work could be done.

7.3.6 **Sprint**

A Sprint is a regular, repetitive work cycle in scrum methodology during which work is completed and made ready for review. It is considered as a basic unit of scrum development and duration of every sprint is defined in advanced. In most of the cases duration is one week to one month.

All scrum sprints are led by a sprint planning meeting where the sprint tasks are established and identified, and an estimated commitment of sprint goals is made. The product owner and Scrum master decide what needs to be moved from product backlog into sprint backlog. A meeting, called daily stand-up meeting, is held during scrum sprint to check the project status and discuss any hurdles and solutions to this. A scrum sprint is followed by a sprint review, where the process is reviewed in order to identify lessons that can be used to improve the next sprint. A sprint retrospective meeting follows the sprint review. This meeting reflects on how work was done during the sprint period. It gives the team a chance to discuss the sprint and think of better alternatives to do things efficiently. At the end of the sprint, the team presents its completed work to the project owner and the project owner uses the criteria established at the sprint planning meeting to either accept or reject the work.
7.3.7 Backlog Grooming

When Product owner submits new user stories to the Backlog then it is the responsibility of scrum master to refine or groom these stories, sometimes to break the larger user stories to smaller stories. The time involved in these activities is called backlog grooming or story-time. These meetings are not longer than one hour and it is something related to breaking larger stories to smaller stories.

7.3.8 Artifacts

7.3.8.1 Product Backlog

The product backlog represents the list of all the requirements of a product and these requirements are submitted by the product owner in the form of user stories. A typical Scrum Product Backlog consists of the following types of items:

- **Features** - It represents the risks, priority, business values assigned to every user story present in the backlog.

- **Technical Work** – It contains the actual work which is to be done on a user story for the development of the project.

- **Knowledge Acquisition** – All the knowledge associated from the requirements to user stories is present in the backlog in form of it.

Everyone in scrum team can edit the product backlog but it is the responsibility of the product owner to order the stories in the backlog for the Development team.

7.3.8.2 Sprint Backlog

Sprint Backlog stores the information on daily basis as it has the list of all the work which is to be addressed by the scrum team in the next sprint. The stories which come to sprint backlog are divided into tasks by the development team. This activity makes the development team to understand what exactly they have to do and in how much time. The process of task assignment in Scrum is like anyone can pick a task and starts working on it. Nobody is assigns any task to others.
7.3.8.3 Burn Down Chart

This is the chart which shows the remaining work in the sprint backlog. It is updated by the Scrum Master on daily basis and then he checks whether the progress report is matching the specified timelines or not.

7.4 Conceptual Schema of AIR

The conceptual model is essential in understanding the overall organisation and conceptual dependency of different entities identified from requirement analysis. Figure 7.1 represent the conceptual model of the AIR with some basic constructs of the model.

7.5 Overall Schema of AIR

This scheme tells us the overall structure and different parts of the model it is also helpful in understanding end to end process of the model (cf. Figure 7.2). After understanding the concept of the model, overall schema helps us in understanding each and every construct of the model. Model in divided into different parts, all of which are discussed below.

7.5.1 Online Knowledge Management

This component of Scrum Development is used to manage the knowledge at every step. This management of knowledge can be offline or online. It varies from organization to organization. When someone wants to use it, firstly he/she will lock it and then use it. No concurrent users can edit it at one time. If one person is using it in editable mode then others can open it in read only form. It has three components

- **Requirements Management** - Product Owner provides requirements to the Scrum Master which was given by the Client. Scrum Master grooms these requirements and sends it to the Product Owner for approval. Requirements of all these steps are maintained in the KM.

- **Discussions** - When new requirements are submitted, these are stored in it and then discussions regarding skill set of all the members and requirements are done.
Figure 7.1: Conceptual Schema of AIR
Figure 7.2: Overall Schema of AIR
At the end of the discussions Scrum Team members assign the user stories among themselves. The log of all these discussions is maintained in Knowledge Management.

- **Backlog** – This is the component of KM where all the information is stored in the form of backlogs. After each Sprint, all the information like who developed the module, functionality and role of the module in the product is saved in Sprint Backlog which is the part of this main Backlog.

### 7.5.2 Task Processing

In Scrum Development, the basic unit is Sprint. One product consists of multiple Sprints. One Sprint has one full complete task to perform which has following steps:

When a user story is assigned by a Team member to himself then that story becomes a task which he has to complete in the estimated time. Each member of the team has to provide estimated time, hours per day and everyday’s work entry is done in the worklog. This task processing is subdivided into four parts:

#### 7.5.2.1 Development

This phase leads to the actual development of the product. This consists of coding of the module which is done by development team members.

**Add Assignee** – For development of every module is assigned to a team member.

**Add Estimate Hours** – For each module’s development and testing, total hours are estimated after its assignment to any member. These are stored in the worklog.

**Add Task Worklog** – For each task, a worklog is maintained in which time needed to complete the task and daily progress of that task is entered.

#### 7.5.2.2 Testing

After completion of coding, one module is sent for testing. Following are the two types of Testing that are done:
Functional Testing

Functional testing verifies that each function of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing and it is not concerned about the source code of the application. Each and every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.

Acceptance Testing

User acceptance is a type of testing performed by the client to certify the system with respect to the requirements that were agreed upon. This testing happens in the final phase of testing before moving the software application to market or production environment. The main purpose of this testing is to validate the end to end business flow.

7.5.2.3 Impediment

If any of the team members is facing an issue while development and testing then it is the responsibility of the Scrum Master to remove that obstacle and provide environment for the work. For this, he/she does the following activities:

**Burn down Charts** -Scrum Master generates Burning Charts on daily basis by calculating the remaining hours. With these charts, everyday’s progress is shown to the whole team.

**Re-assign Stories** –If the estimated time left is more than the actual remaining time then these stories are re-assign to some other member. For this he calculates the remaining hours and efforts.

**Take Corrective Actions** – If any user story is not completed in the current iteration then Scrum Master can add this story to the next iteration or if any member is going on leave then Scrum Master can re assign the story to any of the member.
7.5.2.4 Documentation

After completion of every module, the knowledge is saved in the Online Knowledge Management in the form of documents. These documents are of current iteration, as well as of previous (if any) and next iteration.

7.6 Designing the Agile Information Radiator

After conducting several rounds of discussion and interviews with industry people, we finalised the requirement for design. After understanding the requirements and conceptual working of the tool, next step is to prepare design of the tool (AIR). Unified Modeling Language (UML) based modeling technique is used to build the design specifications. The Unified Modeling Language allows developers to create a model of a software system similar to blueprints of a house. To keep an overview of the whole model, the UML provides several diagrams. Each diagram shows only a few aspects of the system and omits all that are irrelevant to this specific perspective. Together, these diagrams form a complete picture of the model with all its aspects.

7.6.1 Use Case Diagram

A use case describes a set of activities of a system from the point of view of its actors which lead to a perceptible outcome for the actors. A use case is always initiated by an actor (cf. Figure 7.3). In all other respects, a use case is a complete, indivisible description.

7.6.1.1 Actor

An actor is an entity located outside the system that is involved in the interaction with the system described in a use case. An actor may be a person, e.g. a user, but may also be another technical system, e.g. SAP, the operating system, etc. Four actors play vital role in our model. Following are the list of all the actors with their responsibilities:

Product Owner

Product owner is responsible for the creation of product, requirements. He adds the requirement in the backlog and also provides these to the Scrum Master. Product owner
approves the refined document of requirements sent by the Scrum Master. It is the responsibility of product owner for creation of team and release plan.

**Scrum Team**

Scrum team is cross-functional and self-organising. Team reviews the user stories submitted by the scrum master, based on which team develops task to be performed. Task can be development, testing, documentation. Each resource allocates stories to him based on his skill set. This is done under the monitoring of Scrum Master. Self assignment of task is based on the points allocated to the story. Velocity of the team has been calculated and task entries have been updated. At the end of every day, each member puts his/her work’s entries into the log sheet which is stored in the backlog.

**Scrum Master**

Scrum master assigns priorities to the user stories, grooms user stories and creates burn down charts on daily basis, to check if there is any mismatch in milestones set at the start of sprint and milestone achieved. If risks are faced by any of the members of Scrum Team then Scrum Master tracks those risks and tries to find some solution. He can discuss these risks with Product Owner also if any changes are required in the user stories. If burn down charts shows that time required to complete any task is more than the estimated time or if he feels some discrepancies in the self-task assignment process then he can Re-assign the stories to other members.

**Application Administrator**

Application Administrator is responsible for the creation of users from the pool of resources which are available for the project. Other responsibilities of admin are editing user details and making user inactive.
7.6.2 Class Responsibility – Collaborator Cards

Class Responsibility – Collaborator cards help to identify and specify objects or components of an application in an informal way – especially in the early phases of software development. A CRC card describes a component, an object or a class of objects. The card consists of three fields:

- Name of the component (Class)
- Responsibilities of the component
- Names of other collaborating components

As per Use case diagram, mainly there are four components or classes and through CRC cards we have shown the responsibilities and collaboration of each component with other components. How one component is interacting with other components and what is the role of one component in the whole software.
7.6.2.1 Product Owner

The first CRC card is for class Product Owner and its Collaborator is Scrum Master. Here Product owner is performing following responsibilities with the collaboration of Scrum Master (cf. Table 7.4).

- Creates Product – Product owner initiates a new product. He explains expectations from the product to the scrum master.
- Adds Requirement – If any additions are required in the current release of the product then product owner does this.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Owner</td>
<td>Scrum Master</td>
</tr>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>Creates product</td>
<td></td>
</tr>
<tr>
<td>Adds requirement</td>
<td></td>
</tr>
<tr>
<td>Approves Requirement</td>
<td></td>
</tr>
<tr>
<td>Convert requirements to user stories</td>
<td></td>
</tr>
<tr>
<td>Freezes requirements</td>
<td></td>
</tr>
</tbody>
</table>

- Approves Requirement – if client adds some requirements then these are approved by product owner.
- Convert requirements to User Stories – As scrum master has to provide user stories to the scrum team so product owner takes requirements from the client and converts these into user stories and passes to the scrum master.
- Freeze Requirements – When a new release is about to be ready for the production at that stage product owner freezes requirements.

7.6.2.2 Scrum Master

The second CRC card is for class Scrum Master and its Collaborators are Product Owner and Application Administrator. Scrum master has following responsibilities which he completes with the help of product owner, application administrator and scrum team (cf. Table7.5).
• Review Backlog – Backlog is a kind of share point where all the requirements and every kind of documentation is placed for each iteration of the product. Scrum master has the responsibility to inspect its progress at every stage of product development.

• Assign Priorities to Backlog - Scrum master prioritize the requirements present in the backlog according to the release.

• Review User Story Assignments – He reviews the assignment of user stories to the scrum team members. These user stories are provided by the product owner to the scrum master.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrum Master</td>
<td>Product Owner</td>
</tr>
<tr>
<td></td>
<td>Application Administrator</td>
</tr>
<tr>
<td></td>
<td>Scrum Team</td>
</tr>
</tbody>
</table>

**Responsibilities**
- Review backlog
- Assign priorities to backlog
- Review user story assignments
- Monitor tasks
- Create burndown charts
- Track project risk
- Re-assign stories

• Monitor Tasks – Scrum master monitors every task’s progress time to time to measure the development of product.

• Create Burndown charts – Scrum master creates charts to see the discrepancies of task’s allocated time and actual time. If needed, he can re-assign user stories to deliver the product on time.

• Track Project Risks – To deliver the product with expected quality, scrum master tracks risk at each iteration level and takes action to minimize these risks.

• Re-assign stories – If development of product is lagging the expectations then scrum master can re-assign user stories to some different teams.
7.6.2.3 Scrum Team

The third CRC card is for class Scrum Team and its Collaborator is Scrum master. Scrum team performs the following responsibilities by collaborating with Scrum master (cf. Table 7.6).

a) Review User Story – Assigned stories are reviewed by every member of scrum team.

b) Estimate Hours for User Story – After reviewing the user stories, members of scrum team make estimation time plan for assigned story and provide these to scrum master.

c) Assign Resource to User Story – When scrum master handovers the user stories to scrum team the members of the team assign resource to a user story according to their skill set.

d) Create Task – The members of scrum team produce tasks from the user stories.

e) Adds Daily Worklog Effort – At the end of day, efforts given in the product development are entered in worklog sheet by every member. This sheet provides the progress of product on daily basis.

f) Task Completion – While maintaining all the logs related to product, members of scrum team complete their tasks in the given time.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrum Team</td>
<td>Scrum Master</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Review user story</td>
<td></td>
</tr>
<tr>
<td>Estimate hours for user story</td>
<td></td>
</tr>
<tr>
<td>Assign resource to user story</td>
<td></td>
</tr>
<tr>
<td>Create tasks</td>
<td></td>
</tr>
<tr>
<td>Add daily worklog effort</td>
<td></td>
</tr>
<tr>
<td>Task completion</td>
<td></td>
</tr>
</tbody>
</table>
7.6.2.4 Application Administrator

The fourth CRC card is for class Application administrator and its Collaborator is Scrum master (cf. Table 7.7). Application administrator performs only admin tasks asked by scrum master.

a) Create Users – If anyone from the product team needs access to some new application or software, the application administrator provides access to them with the approval of scrum master.

Table 7.7: CRC Card of Application Admin

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create users</td>
<td>Scrum Master</td>
</tr>
<tr>
<td>Edit user details</td>
<td></td>
</tr>
<tr>
<td>Inactive users</td>
<td></td>
</tr>
</tbody>
</table>

b) Edit User Details – If any amendments are required in the roles of team members, these are also done by application administrator.

c) Inactive users – When the task assigned to any member of the scrum team comes to completion state, application administrator inactivates the role of that member from that particular application.

7.6.3 Class diagram

Class diagrams capture the static view of the system and include classes, collaborations, and relationships. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modeling and detailed design modeling. Association Class diagrams might show class hierarchy, inheritance, and realization (cf. Figure 7.4). The static view mainly supports the functional requirements of a system – the services the system should provide to the end users. When we design a Class diagram, it includes following four main elements:
• **Classes**: A class is a representation of an object and, in many ways, it is simply a template from which objects are created. Classes are the main building blocks of an object-oriented application.

• **Attributes**: Attributes are the properties of a class. It is a typed value attached to each instance of a class.

• **Responsibilities**: Responsibilities are the methods or operations performed by each instance of the class.

• **Associations**: Association defines how the objects of one class are related to the objects of the other classes.

After going through the entire requirement document which has been made after extensive discussions with experts and professional, some prominent classes and packages have been identified. In this section we shall present class level view of the tool. Overall, nine classes have been identified.

7.6.3.1 **Product Class**

When a new Project or new release of the project is initiated then the first basic class which contains the detailed information of this project or release comes under this Product class. Product Class has four elements:

7.6.3.1.1 **Class**: Product is the name of class.

7.6.3.1.2 **Attributes**

• **ProductId**: This is the identity number of the product which is being developed. It is the unique key which differentiates it from the other products. For its entire life product is known by its identity number assigned here.

• **ProductName**: It is the name given to the product. Both the name and id are used to synchronize between name and item.

7.6.3.1.3 **Responsibilities**

There are three methods defined for Product class.
• **AddProduct** : AddProduct function will be used to add a new product under consideration.

• **EditProduct** : If any minor changes are required in the current release then these are performed by EditProduct function.

• **DeleteProduct**: As name suggested delete product is used to delete the product from tool e.g. after the completion of project, the product can be deleted from the tool.

• **GPA**: This function stands for general product availability function. This function is used when product is ready to be delivered to the client.

• **Validation**: To invoke the beta testing, this function is invoked to perform testing at client end.

7.6.3.1.4 **Associations**

The Product class has relationship with the following classes.

• **Team Class**: To assign the various operations among the human resources, Product class is associated with Team class.

• **Release Class**: To assign unique numbers to each release of the product, this association is required. One product has multiple releases.

7.6.3.2 **Release Class**

Release class will be used for managing the releases of the product. As agile suggests delivering frequent working software, a product gone through many intermediate releases until final release has been done. Approximately at the end of iteration, a release is planned. Class elements have four elements.

7.6.3.2.1 **Class**: Release is the name of class.

7.6.3.2.2 **Attributes**

• **ReleaseId**: Every release will be assigned an identification number. Id is used for future references to that particular release.

• **ReleaseName**: Like separate id, every release will be assigned a name.
7.6.3.2.3 Responsibilities

There are three methods defined for Release class.

- **ReleasePlan**: This function is used to add a new release in the product when the customer demands some major changes in existing one.

- **EditRelease**: If any changes are required in the current release then this method is used.

- **DeleteRelease**: DeleteRelease will be used to delete the history of a particular release from tool.

- **GetProduct**: This function contains the ProductId number to which the current release is associated.

7.6.3.2.4 Associations

The Release class has relationship with the following classes:

- **Sprint Class**: As every release has many number of iterations, so these are attached by ReleaseId and SprintId.

- **Product Class**: A product can have single or multiple release. Release is associated with a particular product id.

7.6.3.3 Sprint Class

Core of the tool is iteration. Product has releases and every release is composed of none or multiple iterations called sprints. This is the basic unit of development process. Iterations are time boxing in nature. Four elements of this class are:

7.6.3.3.1 Class: Sprint is the name of class.

7.6.3.3.2 Attributes

This class has following attributes:

- **SprintId**: Like product and release, iterations also have a unique identification numbers which are used to recognize iterations. If we have to identify some particular developing component of the product then it will be identified as <ProductId><ReleaseId><IterationId>.
• **SprintName**: It is used to recognize what is being done in that specific iteration. **OwnerID**: It contains the Identification number of that member of Scrum team who is working on that particular iteration.

• **Status**: This attribute has the information regarding the status of the iteration, such as is it completed, in process or just initiated.

• **StartDate**: This attribute will store the starting date of the iteration

• **EndDate**: date: It will store the date when iteration is going to finish

### 7.6.3.3 Responsibilities

There are four methods defined for Iteration class.

• **SprintPlan**: Each release can have many sprints. Each sprint work on some user stories. Sprint plan deals with the work that will be carried out in that particular sprint.

• **EditIteration**: If any updation is required in the process of iteration, this method is used.

• **ApproveSprint**: Acceptance criteria is approved before starting the sprint and after the completion of each sprint work done in sprint is again approved form the client.

• **DeleteIteration**: This method will be used to delete a particular iteration from the product.

• **Verification**: In verification, acceptance testing of sprint is done by scrum master in order to examine that all sprint acceptance criteria has been met.

• **SprintDemo**: After the completion of each sprint, team demonstrates the outcome of the sprint to the client for acceptance or rejection of the sprint.

• **GetRelease**: This function contains the ReleaseId number with which the current iteration is associated.
7.6.3.3.4 Associations

The Release class has relationship with the following classes:

- **WorkItem Class**: Sprint is the outcome of all the work done in work item class. Work item class deals with the task in the sprint.

7.6.3.4 Team Class

For the development of the project, a team should be there. Agile recommends self-organizing and cross functional team. So team in agile manages its role and responsibilities by itself. Roles are also rotated among the team members.

Four elements of this class are:

7.6.3.4.1 Class: Team is the name of class.

7.6.3.4.2 Attributes

This class has following attributes

- **TeamId**: It has the identity number of the team who is working on some particular Release.
- **TeamName**: It is the name assigned to a particular team.
- **ExpectedVelocity**: velocity is calculated by counting the number of units of work completed in a certain interval. So from previous track, expected velocity is calculated for the team.
- **HoursperDay**: This attribute contains the number of hours for which a team is working per day. It tracks the working time of each member of a team.
- **Product**: It is the product in which team is working.

7.6.3.4.3 Responsibilities: There are total three methods defined for Team class.

- **NewTeam**: This method is used to create a new team.
- **EntryinKM**: This method is used for entering the updated documents of everyday work in the Knowledge Management system.
- **GetProduct**: It is the Id of the product for which team is working.
7.6.3.4 **Associations**: The Team class is associated with the following classes:

- **User Class**: The team is created from the user database only. So User class works as an input for the Team class.

- **KnowledgeManagement Class**: To update documentation according to the daily basis work, scrum team members access this class.

- **Requirements Class**: Team class is directly associated with user class for creation and finalisation of requirement. Scrum team include product owner, scrum master and development team.

7.6.3.5 **User Class**

This represents a pool of people from which team members are pulled according to their skill-set to perform on particular user stories. As in agile, requirements are amended with the user stories before development starts.

Four elements of this class are:

7.6.3.5.1 **Class**: User is the name of class.

7.6.3.5.2 **Attributes**

This class has following attributes:

- **UserID**: In the pool of users, every user has one unique Identification number which differentiates it from the other users. When one user is selected for a particular team then this UserId is used for the association of User and Team classes.

- **FirstName**: It holds the first name of the user.

- **LastName**: It holds the last name of the user.

- **Email**: To store the email id of the user.

- **Title**: To store the title of the user.
7.6.3.5.3 Responsibilities

There are two methods of class users:

- **AddUser**: This method is used to add a new person to the pool.
- **DeleteUser**: This method is to delete a particular person from the pool.

7.6.3.5.4 Associations

The Product class has relationship with the following class:

- **Team Class**: The team is created from the user database only. So user class works as an input for the Team class.

7.6.3.6 WorkItem Class

This class is used to work on the request which we get in the way of user story. Work item is a story which has been assigned iteration and at the end of iteration, work item is converted to intimidate deliverable piece of software. This class defines all the characteristics of work item which has been chosen from requirements. Four elements of this class are:

7.6.3.6.1 Class: Product is the name of class.

7.6.3.6.2 Attributes

This class has following attributes:

- **WI_Owner**: It holds the name of the person who generates the request or owes the Work Item.
- **WI_Resolver**: This attribute holds the name of the person who resolves the issue related to some particular work item.
- **StoryPoints**: Each user story are assigned points according to its complexity of it. Each organisation has its own criteria for assigning complexity points to a user story.
- **ActualHours**: Time taken by the user for completion of task.
• **EstimateHours**: This is time period allocated for a particular work item, which means time period allotted to the story for its completion.

• **Startdate**: It stores date when work on user story has been started.

• **Completiondate**: It stores the plan date on which some particular work item has been completed.

• **Status**: It is used to store the status of the work item whether its completed, in process or initiated.

• **Type**: Type describes, the category to which the task belongs to. Is it development, testing, impediment or documentation?

• **RequestId**: To which request the present work belongs to, is expressed by request id.

• **Summary**: This attribute has brief information about the work item.

• **Description**: It contains the details and type of association of particular work item with other classes.

### 7.6.3.6.3 Responsibilities

Further methods have been included in the class that enhances its working:

• **EffortEstimation**: Effort estimation means how much effort per person is required to complete the task. Estimation of efforts is done on the basis of velocity of the team.

• **TaskEstimation**: Task estimation is to know how much working hours are required for the completion the task. Task estimation is done on the basis of story points assigned to the task according to its complexity.

• **GetRelease**: This function contains the ReleaseId number to which the work item is associated.

### 7.6.3.6.4 Associations

• **Request Class**: Work item class works on the user stories generate by request class. Requests are assigned to differ user according to the workitem.
7.6.3.7 KnowledgeManagement Class

The role of this class in the whole process is to maintain the documentation of each release and iteration. This class is kept updated by the members of scrum team only. How one organization is developing its overall product and through what phases it needs to pass is described by this class only. The four elements of this class are:

7.6.3.7.1 Class: KnowledgeManagement is the class name.

7.6.3.7.2 Attributes

This class contains the following attributes:

- **SprintID**: This attribute is to specify the sprint id for which documentation is done.
- **Date**: It contains the date on which updations are done.
- **TeamID**: contains the ID of the team who is entering the data.
- **UserId**: It contains the ID of the user who is entering the data.
- **CheckoutTime**: It specifies the time at when some user starts entering data on the sharepoint.
- **CheckinTime**: It specifies the time at when some user stops entering data on the sharepoint.
- **Document**: This is of type ProcessModel. This contains the information which is the outcome of knowledge sharing process.

7.6.3.7.3 Responsibilities

There are four methods defined for KnowledgeManagement class:

- **Offline**: When some team member is updating documents offline on his/her own machine then it is called offline method.
- **Online**: If team member is entering data on some sharepoint or on some intranet site then this method is called online.
- **Checkout**: This method is used to reserve the common document for updates so that not more than one member can use it in parallel.
• **Checkin**: This method is used to release the common document after completing the updates.

### 7.6.3.8 Request Class

This class explains the basic element of the whole process. When we start developing new product then request is generated at first and it is developed from the user stories and epics. This class consists of two classes: UserStory Class, Epic Class.

The four elements of this Request class are:

#### 7.6.3.8.1 Class: Request is the name of the class.

#### 7.6.3.8.2 Attributes

This class contains following attributes:

- **RequestId**: This attribute contains request number which is then specified in the iteration number.
- **RequestName**: This attribute contains the name of the request.
- **RequestStateId**: It stores the state of the request that it is submitted or completed.
- **RequestPriorityId**: This field contains the priority of the request e.g High, Low, Medium.
- **StartDate**: It contains the date on which request is submitted.
- **EndDate**: It contains the proposed end date of the request.
- **OwnerId**: This attribute contains the id of the owner who submits the request.

#### 7.6.3.8.3 Responsibilities

There are total two methods defined for Request class:

- **GetIteration**: This method is used to get iteration number for some particular request.
- **GetRelease**: This method is used to get Release number for some particular request.
7.6.3.9 UserStory Class

This class is used to provide user stories to the request class. User stories are basically requirements from the product owner. The four basic elements of this class:

7.6.3.9.1 Class: UserStory is the class name.

7.6.3.9.2 Attributes: This class contains the following attributes:
  - **UserStoryID**: A unique identification is assigned to user story.
  - **UserStoryName**: to identify user story along with id, name is assigned to it for easy remembrance.
  - **UserStoryDesc**: It contains the description of user story.

7.6.3.9.3 Responsibilities

  - **DeleteUserStory**: It is user-defined function which is used in this class to delete any user story.
  - **GetUserStory**: This function is used to generate user story from a request.
  - **ScrubUserStory**: Scrubbing is used to polish a user story according to actual need of the request. In scrubbing user story is minimize to single functionality.

7.6.3.10 Epic class

This class represents historical data which may be helpful to derive user stories.

7.6.3.10.1 Class: Epic is the class name.

7.6.3.10.2 Attributes

  - **EpicID**: It contains one unique number for each epic.
  - **EpicName**: It contains name of the epic.
  - **EpicDesc**: It defines the epic.

7.6.3.10.3 Responsibilities: This class has the following responsibilities:

  - **DecomposeEpic**: This user-defined function is used to decompose epic into user stories.
Figure 7.4: Class Diagram of AIR
7.6.4 Knowledge Management Process

The working of knowledge management process describes in figure 7.5 is further explained with the help of elaborated class diagram. This diagram shows the process of knowledge sharing among all the team members and entities. KM process class describes how knowledge is maintained and transferred in the project. The final information is stored in the Backlog which works as the repository for further releases and change requests. Below is the information for all of its classes and their roles.

7.6.4.1 Knowledge Resource

This is the class actually used to retain the information in form of documents. Four elements of this class are:

7.6.4.1.1 Class: Knowledge Resource is the class information.

7.6.4.1.2 Attributes

This class has following attributes:

- **Author:** It contains the name of the person who created/uploaded the document
- **Date:** It holds the date on which document is created.
- **Version:** When some new update is done to the document, the document should be saved with a new version like version 1, 1.1, and 1.2 and so on.
- **Status:** It saves the current state like document is in submitted state or approved, checked out by someone (locked) etc.
- **Objective:** It represents the objective of the document. Most of the time it is one line statement about the document.

7.6.4.1.3 Responsibilities

This class performs the following two methods:

- **addDocument:** This function is used to upload document on the sharepoint or some common place where all the documents of the project are saved.
• **deletedocument**: This function is used to delete document from the sharepoint.

Except this class, the knowledge sharing process is showing specialization and Distribution among various elements. Below are the details of the elements:

### 7.6.4.2 Knowledge

This is the basic unit of the document which has relevant information about some iteration or release.

- **Description**: It refers to the description about the actual documentation.
- **Knowledge_list**: It holds the index number for all the documents present in the backlog.

### 7.6.4.3 KnowledgeElement

This defines the document description properties.

- **Description**: It works like metadata for any particular document.
- **Impact**: How some particular knowledge is going to impact the information repository.
- **Criticality**: It defines the criticality like severe, major or minor.
- **Type**: It holds the data type of the information.

### 7.6.4.4 Process

It defines the properties of process from which knowledge is generated.

- **Objective**: Objective can be checkout process, check-in process, or approval process
- **Description**: It holds the description that what the process does in whole knowledge sharing process.

### 7.6.4.5 ProcessModel

Process model states the overall description of process.
• **input_knowledge_element**: It tells the information about the process or metadata about the process.

• **input_knowledge_application**: It holds process definition or actual code.

• **output_location**: It holds the location details where the file will be saved.

### 7.6.4.6 KnowledgeApplication

This is the actual code to access the documents.

• **application_code**: It is the actual code which is being used to access the data.

• **code_author**: It tells about the owner of the code who has written the code.

• **code_language**: It tells the language in which the application code is written.

### 7.6.4.7 Case: case_list

It controls the versions for documents. Different version of the document are made and maintained with the help of this class.

### 7.6.4.8 Case_report

It represents the report attributes and formats of the report.

• **Input_knowledge_element**: It represents the data type of knowledge element which has been created.

• **Input_knowledge_application**: It tells the application of knowledge element which has created and stored in AIR knowledge base.

• **Input_file**: This file represents action to be perform with Knowledge Resource.

• **Key_parameter**: Key parameters are used to search a particular knowledge element form the AIR knowledge base.

### 7.6.4.9 Discussion

This is the application used by the team members to interact with each other.

• **Chatlist**: It contains the list of all the available members for chatting.
Figure 7.5: KM Process of AIR
The main purpose of a sequence diagram is to define event sequences that result in some desired outcome. The focus is less on messages themselves and more on the order in which messages occur; nevertheless, most sequence diagrams will communicate what messages are sent between a system's objects as well as the order in which they occur. Following are some of the objects of sequence diagram (cf. Figure 7.6).

**Client:** Client is the person who plays the ancillary role in the scrum development. He is involved in some of the activities like requirement submission and at the end when product comes in the state of testing.

**Product Owner:** Product owner plays a core role in the scrum development. He has an eye on the complete process of development, starting from requirements till product release.

**Scrum Master:** Scrum master also has core responsibilities throughout the product development. He is committed to every activity of the product development.

**Scrum Team:** Scrum Team also plays a vital role in the actual implementation of the product. They assign tasks to themselves and then start working according to the requirements.

Following are the sequence steps of Scrum Development starting from Requirement submission to Product Release:

1. When a new product or any Change Release comes the Client initiates a product and requirements in the backlog.

2. Client submits the requirement to the Product Owner

3. Product Owner provides these requirements to the Scrum Master.

4. Scrum Master does grooming (refinement) of these requirements and makes a final document of the feasible requirements.

5. He submits this final document to the Product Owner.

6. Product Owner approves this final document and sends back to the Scrum Master.
7. Scrum Master adds this document to the backlog.

8. At this time when requirements are finalise; Client and Product Owner prepares a Release Plan which consists of the details that what actually need to be completed and in how much time.

9. As Release is combination of multiple Sprints. So from Release plan Sprints are derived which are short term releases.

10. Then this Sprint Plan is sent to the backlog.

11. Team takes the user story from the backlog.

12. These user stories are scrubbed by scrum master and team for refinement of team.

13. Team submits the acceptance criteria of the sprint to the product owner.

14. After inspection, Product owner approves the acceptance criteria of the sprint.

15. Now the discussion happens between Development Team members and Scrum Master for the estimation of time and the assignment of different tasks among team members.

16. After discussion, task assignment is done by the team members by themselves according to their skill set. This all assignment is monitored by the Scrum Master.

17. Document generated in the sprint is updated into the backlog.

18. After completion of the Sprint, acceptance testing of that sprint is done by the Scrum Master and Team Members.

19. After the successful testing of the sprint, Demo is given to the Client at the completion of each Sprint. Sprint starts from getting user story from backlog and end at this point. There can be multiple sprints in single release.

20. At this point, Product is availed at client’s site.

21. Here testing is done for that particular Sprint. This type of testing is called Beta Testing.

22. If any issues are faced at client’s site, these are reported to the Backlog by the Product Owner so that these must be taken care in next Sprint.
Figure 7.6: Sequence diagram of AIR
7.6.6 State Chart

A state diagram shows a sequence of states an object can assume during its lifetime, together with the stimuli that cause changes of state. A state diagram describes a hypothetical machine (finite automaton) which at any given time is found in a set of finite states. State diagram is divided into different sub machine states to describe major states in the development process. Further sub-machine states are divided into finite states (cf. Figure 7.7).

7.6.6.1 Requirement Gathering

This sub machine state comprises of two states. The submit requirement state is initiated when a new product request is created by the client. In this state, requirements are submitted by client and product owner converts them in legal requirements and product owner groom the requirements. The second sub-state is approving requirements. After grooming scrum master submits requirements for approval to product owner. If product owner disapproves the groomed requirements then requirements are sent back to the scum master for further grooming again. If product owner approves the requirements, these requirements are added into backlog.

7.6.6.2 Backlog Updating

Whenever documents have to be stored in database, backlog updation is used. Documents updated in backlog can be related to requirements, KM, chat logs or information generated during the course of action that can be helpful for future references.

7.6.6.3 Sprint

Sprint is sub-machine state which includes multiple states. Sprint state works in iteration until all the requirements generated are not fulfilled.

Division of Task: In sprint first state is the recognition of task. Tasks include development, testing, impediment or documentation. Exit from this state depends upon finalization of the category of the task.
**Work Distribution**: Agile emphasis on self-organising and cross-functional teams. Self assignment of task has been done by team members. Scrum master is only for monitoring or for reduction of hindrances faced by team. Distribution of task includes effort estimation and task estimation. Effort estimation includes effort required for completion of a particular task. It is calculated as person per month. Task estimation means time and cost required for completion of task. Depending upon effort and task estimation teams generate their velocity (Capacity of doing work). According to velocity, distribution of work has been done.

![State Chart of AIR](image-url)

*Figure 7.7: State Chart of AIR*
**Task Completion Status:** Status of team is monitored by scrum master. Status of team is measured by burned down chart. If team is lagging behind the schedule, scrum master re-assigns stories to different team members. Corrective actions are taken by the scrum master so that project can be on track.

**Verification:** Verification is performance of acceptance tasting as per acceptance criteria set by the client. Once the acceptance criterion is met, one sprint completes. One release can have multiple sprints.

**7.6.6.4 Release**

Once every user story in release plan is finished, product enters into release state. Once release of project is ready, scrum master declares general product availability of project to the client. Here client performs beta testing of the project, called validation.

**7.6.7 Activity Diagram**

Activity diagram is basically a flowchart to represent the flow from one activity to another. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc (Figure 7.8). It captures the dynamic behaviour of the system. In our system activity starts with the generation of requirements from client. Client submits these requirements to the product owner and product owner forwards these requirements to Scrum master for grooming. Scrum master converts these requirements to user stories. User stories are the smallest entity of requirements which cannot be broken further. Scrum Master, after grooming again sends back to product owner for approval of user stories. If product owner approves the user stories then these user stories are added into backlog, otherwise product owner sends them back to scrum master for further grooming of user story. After requirements are added into backlog, product owner creates the release plan. In release plan highest priority user stories are added. Sprint plan is prepared on the basis of release plan. A release plan can have one or many sprints.
Figure 7.8: Activity Diagram of AIR
User stories finalised for a particular release is added into sprint plan. Then these user stories are divided into task by the scrum team. Self assignment of tasks has been done at team level. Each member of team updates the task log on daily basis. Scrum master, on the basis of these logs, develops burn down charts. These charts show the actual status of the project.

If project is lagging behind the schedule then it is the responsibility of scrum master to re-assign user stories to different person so that project can be completed in stipulated time frame. After the completion of the sprint acceptance testing is done. If sprint/project pass acceptance testing, sprint demo is performed at clients’ end and at the end of the project product, is released to the client.

### 7.6.8 Component Diagram

Component diagrams are used to model physical aspects of a system. Physical aspects are the elements like executables, libraries, files, documents etc. which resides in a node. A component is an executable piece of software with its own identity and well-defined interfaces. A distinction should be made between component definitions (e.g. “person”) and component instances (e.g. “Gabby Goldsmith”). Component diagrams show the interrelations between components. In practice, UML components are very similar to packages: they define boundaries, and group and structure a set of individual elements. Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment. A component defines its behaviour in terms of provided and required interfaces. As such, a component serves as a type whose conformance is defined by these provided and required interfaces. One component may, therefore, be substituted by another only if the two are type conformant (cf. Figure 7.9). Larger pieces of a system’s functionality may be assembled by reusing components as parts in an encompassing component or assembly of components, and wiring together their required and provided interfaces (cf. Figure 7.9).

A component is modeled throughout the development life cycle and successively refined into deployment and run-time. A component may be manifested by one or more artifacts, and in turn, that artifact may be deployed to its execution environment. A
deployment specification may define values that parameterize the component’s execution.

### 7.6.8.1 Wamp Server

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database and also helps to manage the databases. It has following responsibilities in one whole system:

- Manages MySQL services
- Provides access to local host or everyone
- Manages your servers’ settings
- Access your logs

### 7.6.8.2 Web Browser

A web browser or browser is a software application used to retrieve, present and traverse information from World Wide Web. An *information resource* may be a web page, image, video or other piece of content. Hyperlinks present in resources enable users to easily navigate their browsers to related resources. A web browser can also be defined as an application software or program designed to enable users to access, retrieve and view documents and other resources on the Internet.

### 7.6.8.3 Home.php

PHP is a server scripting language, and is a powerful tool for making dynamic and interactive Web pages. Through these web pages, one can interact with the whole application.

### 7.6.8.4 Database Server

Database server is the term used to refer to the back-end system of a database application using client/server architecture. The back-end, sometimes called a database server, performs tasks such as data analysis, storage, data manipulation, archiving, and other non-user specific tasks. Here Database server is used to manage all the documents.
which are updated after each process; it also contains the information about the team member’s database as well as logs of all the activities.

Figure 7.9: Component Diagram of AIR

7.6.9 Deployment Diagram

Deployment diagrams show a system's physical layout, revealing which pieces of software and hardware run. Deployment diagrams show which components and objects run on which node (processes, computers) how they are configured and which communication relations exist between them. A node is an object which is physically present at runtime and has computing power or memory, such as computers (processors), devices, and the like. The Deployment diagram helps to model the physical aspect of an object-oriented software system. It models the run-time configuration in a static view and visualizes the distribution of components in an application. In most cases, it involves modeling the hardware configurations together with the software components (cf. Figure 7.10).
Deployment describes the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams show how they are deployed in hardware.

Figure 7.10: Deployment Diagram of AIR
So most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as:

- Visualize hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe runtime processing nodes.

### 7.7 Verification and Validation of the Tool

An open source UML modeling tool, called StarUML™, for modeling the UML diagrams has been used. StarUML™ strictly adheres to the UML standard specification specified by the OMG for software modeling. This tool helps in drawing UML diagrams and simultaneously validates them also. Tool itself provides mechanism for validation the diagrams. Tool restricts in establishing any invalid relationship which is not syntactically correct. This means all the relationships build are syntactically correct.

Tool also helps in verifying the model. Tool cross-checks all the elements and their relations. StarUML™ automatically verifies the software model developed by the user, facilitating early discovery of errors. StarUML™ includes many useful Add-Ins with various functionalities: it generates source codes in programming languages. Tool also helps in generating documentation of the model, including Design Assessment (DAS), Design Review Document (DRD), PseudoCode(IMP), UseCase Documentation Specification (UCS). Theses diagrams are also cross checked with the recommendations given by Unhelkar (2005). Author has given detailed Verification and Validation checks for syntactic correctness, semantic completeness and consistency and aesthetic quality of the UML diagrams. This study followed detailed checklists for different UML diagrams in order to enhance their quality. Table 7.8 summarises the different checklists used for syntactic, semantic and aesthetic checks of the diagrams. Detailed tables of checklists are given in Tables D1 – D18 in Appendix D
Table 7.8: Checklists for Verification and Validation

<table>
<thead>
<tr>
<th>S. No</th>
<th>Checklist</th>
<th>Appendix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Syntax Checks for Classes (Advanced)</td>
<td>Table D1</td>
</tr>
<tr>
<td>2</td>
<td>Semantic Checks for Classes</td>
<td>Table D2</td>
</tr>
<tr>
<td>3</td>
<td>Aesthetic Checks for Classes</td>
<td>Table D3</td>
</tr>
<tr>
<td>4</td>
<td>Syntax Checks for Class Diagrams</td>
<td>Table D4</td>
</tr>
<tr>
<td>5</td>
<td>Semantic Checks for Class Diagrams</td>
<td>Table D5</td>
</tr>
<tr>
<td>6</td>
<td>Aesthetic Checks for Class Diagrams</td>
<td>Table D6</td>
</tr>
<tr>
<td>7</td>
<td>Syntax Checks for Sequence Diagrams</td>
<td>Table D7</td>
</tr>
<tr>
<td>8</td>
<td>Semantic Checks for Sequence Diagrams</td>
<td>Table D8</td>
</tr>
<tr>
<td>9</td>
<td>Aesthetic Checks for Sequence Diagrams</td>
<td>Table D9</td>
</tr>
<tr>
<td>10</td>
<td>Syntax Checks for State Chart Diagrams</td>
<td>Table D10</td>
</tr>
<tr>
<td>11</td>
<td>Semantic Checks for State Chart Diagrams</td>
<td>Table D11</td>
</tr>
<tr>
<td>12</td>
<td>Aesthetic Checks for State Chart Diagrams</td>
<td>Table D12</td>
</tr>
<tr>
<td>13</td>
<td>Syntax Checks for Component Diagrams</td>
<td>Table D13</td>
</tr>
<tr>
<td>14</td>
<td>Semantic Checks for Component Diagrams</td>
<td>Table D14</td>
</tr>
<tr>
<td>15</td>
<td>Aesthetic Checks for Component Diagrams</td>
<td>Table D15</td>
</tr>
<tr>
<td>16</td>
<td>Syntax Checks for Deployment Diagrams</td>
<td>Table D16</td>
</tr>
<tr>
<td>17</td>
<td>Semantic Checks for Deployment Diagrams</td>
<td>Table D17</td>
</tr>
<tr>
<td>18</td>
<td>Aesthetic Checks for Deployment Diagrams</td>
<td>Table D18</td>
</tr>
</tbody>
</table>

7.8 Advantages/ Features of Proposed AIR

AIR proposed in the study has the following unique features.

1. The model has been described in an open and general representation schema that can be implemented by any software engineering organisation by customising it in terms of the organisation’s specifics.

2. The model has been designed according to open source technologies to be cost effective. Organisation can customise the proposed tool and modify it according to their specific needs.
3. UML modeling techniques are used to model different aspects of the tool. These diagrams state different behaviours of the tool. These diagrams help the organization for better understanding and easy implementation of the tool.

4. Two schemas of the tool have been presented for the convenience of software engineering professionals. Conceptual schema describes the major components of the system and their characteristics. Overall schema gives detailed information of the components, their connectivity and flow of control in the tool. These two schemas give freedom to the organizations to enhance the tool according to their need.

5. Model has been based on standard and universally accepted notations of UML language. SatrUml tool is used for designing of the tool, it helps in building syntactically and semantically accurate tool.

6. Tool is tested for its quality. Tool is cross checked with the checklists provided by Unhelkar (2005). These checklists help in enhancing the quality of the proposed tool.

7.9 Future Scope of the AIR

The AIR can be extended in future to cover more and more parts of a software engineering organisation and to provide support for more software engineering activities. The AIR can be extended by widening its application scope by adding other kinds of artifacts into its knowledge base like:

- Lessons learnt about software engineering objects or technologies
- Measurement data
- Project documents generated
- Online guidance of agile software engineering activities
- Reference materials
- Reusable artifacts to support knowledge usage
- Yellow pages

AIR can be extended in near future to include some more agile practices according to need of the organisation which intends to implement the tool. By adding more agile practices, the tool can be generalized for all agile practices.