An environment for open user innovation in an airport setting

Software for Use by Larry L. Constantine & Lucy A.D. Lockwood
## Table Of Content

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Define Users</td>
<td>4</td>
</tr>
<tr>
<td>Real users and others</td>
<td>4</td>
</tr>
<tr>
<td>Collaborative requirements dialog</td>
<td>5</td>
</tr>
<tr>
<td>Face to face interview</td>
<td>5</td>
</tr>
<tr>
<td>Goal of system</td>
<td>6</td>
</tr>
<tr>
<td>Requirements of Passengers:</td>
<td>6</td>
</tr>
<tr>
<td>Requirements of Security staff:</td>
<td>6</td>
</tr>
<tr>
<td>Requirements of Terminal staff:</td>
<td>6</td>
</tr>
<tr>
<td>User role model</td>
<td>6</td>
</tr>
<tr>
<td>Operational Context</td>
<td>7</td>
</tr>
<tr>
<td>Operational model</td>
<td>7</td>
</tr>
<tr>
<td>Incumbent profile</td>
<td>7</td>
</tr>
<tr>
<td>Interaction profile</td>
<td>8</td>
</tr>
<tr>
<td>Information profile</td>
<td>8</td>
</tr>
<tr>
<td>Environment profile</td>
<td>9</td>
</tr>
<tr>
<td>Aspect no. 1</td>
<td>9</td>
</tr>
<tr>
<td>Aspect no. 2</td>
<td>9</td>
</tr>
<tr>
<td>Profiles put together</td>
<td>10</td>
</tr>
<tr>
<td>Task modeling and Use cases</td>
<td>10</td>
</tr>
<tr>
<td>Scenarios &amp; Use Cases</td>
<td>11</td>
</tr>
<tr>
<td>Help system and documentation</td>
<td>12</td>
</tr>
<tr>
<td>Usability Inspections</td>
<td>12</td>
</tr>
<tr>
<td>Inspection Method</td>
<td>13</td>
</tr>
<tr>
<td>Collaborative Usability Inspections</td>
<td>14</td>
</tr>
<tr>
<td>Inspection Process</td>
<td>14</td>
</tr>
<tr>
<td>Conformance Inspection</td>
<td>15</td>
</tr>
<tr>
<td>Project Timeline</td>
<td>16</td>
</tr>
<tr>
<td>Advantages &amp; Drawbacks</td>
<td>17</td>
</tr>
</tbody>
</table>
**Introduction**

The process shown by Constantine and Lockwood in their book “Software for Use” and followed in our project, suggests the following structure:

![Diagram of User-centered System Design process]

Though at first hand it might look as a conventional waterfall life cycle it isn’t, the difference comes that on the Usage-centered design from the above diagram, it is a concurrent engineering process where independent activities are done at the same time whenever it feels to be practical. It also isn’t an iterative software life cycle since the activities are only needed to be done once, though as it can be seen each of them interacts with a few of the others.

All the activities can be or can’t be incorporated in the software project depending on the project’s needs and focus, that’s for the ellipses on white background, the ones marked in red are indicated to be always needed in the process since they are the core of the process suggested by Constantine and Lockwood.

The process begins with its core activities: Collaborative Requirements Dialogue, Task Modeling and Domain Modeling. The first one is a conversation and negotiation (or a series of them) between developers and the future users of the future software that wants to be developed, its objective is to extract the requirements of the future system. Second one is focused on obtaining a complete idea of the jobs that need to be supported, this is done through definition of the task models and role models. The last activity being briefly introduced, Domain Modeling, focus on explaining the application domain, this can be done with a typical entity-relationship model or a domain class model. It is going to define the vocabulary and operation of the system.

In the optional activities of the process we have the Interface Content Modeling, which can be done in parallel with the Task Modeling activity. Implementation Modeling is going to define the detailed design of the application and prototyping.
Usability Inspections happen twice in the process, one prior the implementation activities (defined by Architectural Iteration and Concentric Construction) and the other one right after them.

On the sides of our User Centered process we have the Operational Contextualization and Standards and Style Definition activities. The first one focuses on adapting the defined design to the environmental limitations and operational conditions where our future system will be used at. It is really important to center the attention on the fact that it is an activity that evolves a lot through time and thought different activities in the process instead of being dealt with at the very start of the process as more standard processes suggest.

Similar for the Standards and Style Definition activity, these standards and style definitions are established through the construction of many the other activities through time in a concurrent fashion.

Interface Content Modeling, Implementation Modeling, Object Structure Design, Concentric Construction and Architectural Iteration won't be the focus of this document since those activities are more oriented to the implementation task which lacks the importance of the others for our project analysis.

**Define Users**

The development process is a kind of dialogue between developers and users, and both of them need to be involved in the process. Users are the experts on use and developers are expert on development.

Users are the sources of many things that can determine the success or failure of the development. They can supply the information that will help us to identify and understand the basic requirements for the system to be built. They are the sources of feedback on our solution for their problems. They also bring changes in requirements and introduce the troubles into the project.

Since users play vital role in the development process, before we can start designing our system, the top thing we should do is to find who the users are.

**Real users and others**

As the book mentioned, users are divided into two kinds: real users and others.

The users whom we need to reach and understand are the real users of the system, that is, the end users who will have actual hands-on interaction with the software. They are also the main sources of requirements, feedbacks and changes.

In our project, to build an environment for open user innovation in an airport setting, we define the end users as the following three kinds:

- Passengers
- Security staff
- Terminal staff
The others are not the people who use the system as tools. They could be the client, who pays the bill, managers of end users, upper management within the airport organization, technical support and so on.

Here we just focus on the real users.

**Collaborative requirements dialog**

At the top of the activity model in this book is a collaborative requirements dialog. This is the activity that determines, at least initially, what the system to be designed and built must do. Defined requirements are the foundation of later design as well as the basis for quality. (Software for use, P486)

As the book mentioned, settling requirements is neither a matter of simply finding out from users or clients exactly what to build nor a matter of our simply telling them what it is we are going to do for them. (Software for use, P486) The requirements dialog is a specialized process of mutual exploration, conversation and negotiation between developers and their users and clients to establish the requirements of the system to be constructed.

**Face to face interview**

Obviously, users are heavily involved in this activity.

We would interview our users: passengers, security staff, terminal staff and have a talk with them. The discussion and brainstorming process will center on the basic purpose. The following four questions will be the topic of the dialogue.

What is the primary purpose of this system?

How would you expect to use this system?

What would you expect to accomplish with such a system?

What is the reason for doing this particular task?

In the face to face meeting, we are supposed to discuss the above predefined general topics, because it will be more time efficient. But we want to use the free-form and conversational style instead of ticking off a long list of questions to the users. In that case, the meeting seems more like a collaboration and more conducive to a cooperative atmosphere. Otherwise, we are more likely to leave out important asides with asking a lot of questions.

After the interview, the results should be consolidated into a statement of essential purpose.

The followings are our assumptions for the interview results.
**Goal of system**
- Improve the security inside of airport
- Save passengers’ time and make the process more efficient.

**Requirements of Passengers:**
- Multi-touch screen
- Information about their flight: destination, flight No, boarding time, flight time, seat, Gate No, Terminal No
- Delayed flight information
- Luggage information (where are they)
- Information about destination: weather, local time, tourist information
- Map of the terminal (where we are, shopping or restaurant areas...)
- Passport becomes the flight ID instead of having to carry around the boarding card.
  Passengers scan their passports to enter the security area and they are validated with their flight to stay and get around the terminal.
- Be able to print any of the displayed information.

**Requirements of Security staff:**
- Destination of flight
- Flow of passengers

**Requirements of Terminal staff:**
- Flow of passengers

**User role model**
The collaborative requirement dialogue requires both technical and interpersonal skills. The user role model is the technical skill developed from this dialogue.

Mere discussion is not our object. In order to make the information clearer and more correct and incorporate our understanding into the system we built, we need to model what we learn from our users, to confirm with them and with our clients our understanding of the work to be supported.

The process is: listen and learn from users, build a model, share it with users to get feedback and then refine the model.

To build the model, we represent the relationships between users and systems as roles. And we list the user roles which describe the needs, interests, expectations, behaviors and responsibilities. We give each user role a name that shows the basic nature of the role and provide a brief description demonstrating the role’s most relevant characteristics.

The book suggests us to use brainstorming to list candidate roles at the beginning, due to the minimum discussion. Here we already have three candidate roles: passengers, security staff and terminal staff.

Once the initial list of candidate roles has been generated, we are supposed to sort them and fill in the details describing their needs, interests, expectations and behaviors of each role.
And we will first think of one particular user and then generalize toward the more abstract user role. For instance, we start with a passenger in mind and try to conclude his role with the help of interview results from passengers:

The followings are what we have done on the user role model

**Passengers**

First-time use; rapid, easy operation; simple interface; flight information; luggage information; map of terminal; destination weather; scan the passport to enter security area.

**Security staffs**

Frequent use; rapid, easy operation; simple interface; flow of passengers; destination of flight.

**Terminal staffs**

Frequent use; rapid, easy operation; simple interface; flow of passengers.

**Operational Context**

According to the authors it is important that software is fitted to where and when it is used.

“*Truly usable software is highly attuned to its environment.*”

This means that the developers need to keep the operational context in mind when designing a system. This falls under the *Context Rule*. To get an understanding of the context we can follow the books concept of the *operational model*.

**Operational model**

The operational model consists of different profiles that should be considered to get a complete understanding of who will use the system, where it will be used and what it should be used for.

**Incumbent profile**

The incumbent profile tells us what kind of users will use system. In the case of the passenger this will include almost everyone. The users will be ordinary people. They will also considered as novice users. Really frequent flyers might become somewhat familiar with it, but to most it will be something completely new. It will include almost all ages, perhaps not children and youth under a certain age. Youth under 18 years of age cannot travel unaccompanied, so there’s no real reason to try to include them.

In the airport people of many different nationalities pass through every day. Therefore it will be useful to include different languages into the system. Talking with a domain expert and viewing statistics of what nationalities are the most common will give a hint to which

---

1 Page 298
languages are most common. These should be the easiest to select in the system, and maybe the most common one should be set as default. To keep the users from having to select a language for the most common options, perhaps we could consider using a lot of symbols and making the user interface as graphical as possible. Since the user is supposed to scan the passport at the monitor to get appropriate information about his or hers flight, maybe it could automatically change the language to the nationalities official language.

Religion and ethnicity should not affect the understanding of the passenger help system. Colors and symbols have varied meanings around the world, but there is not enough time to do a lot of testing to find the right ones. That’s why it might be useful to incorporate proven international standards into the systems.

The system should also work for people with disabilities, so fuzzy graphics and backgrounds are out of the question. Simple and high-contrast graphics must be used with the design. For visually impaired a discussion of possible audio guidance must be had. Perhaps it will be better to let them have guidance by human staff instead and not include them in this system.

This is for the passengers, although for the system to be used by the security staff much of the same will apply. The security staff comes with different ages, religions and ethnic backgrounds. Perhaps it’s a bit easier to standardize some things though, for example the language. It will not have to be designed for completely novice user every time either. Since we consider this as a very simple device, with a static screen showing the passengers boarding card information, and whether the passenger is heading in the right direction or not, this interface will not need any heavy designing.

**Interaction profile**

As mentioned earlier the system will have to be designed so that it can be used by novice users. Most of the passengers will be novice users if they are not very frequent flyers, and these will probably not be as interested in the system as others as they should already know their way around the airport quite well.

This profile tells us how the user will interact with the system. What will the system be used for? We get some of this information from the requirements dialogue. From this we get information about what the system should be used for by the domain experts. They have knowledge about what passengers needs help with. By using this knowledge we should try to design the interface so that the most common options are easily selectable. If for example the most common option is to see the guidance map to the passengers gate then this should be very easily selected. Making the most common information accessible in as few steps as possible is another goal for the developers. This is speeding up the system for the user.

The security staffs interface should probably not have any interaction. Maybe we will find that a few buttons useful, like for example reset. Other than that scanning the passenger’s passport should be enough to get the proper information.

**Information profile**

The information profile will also evolve from the requirements dialogue. What kind of information will be useful for the user? What kind of information will be effective to
incorporate into the system? We will not want the users to spend more time using this system than to navigate using currently available methods. We also want to limit the information available in the system to prevent people using the kill time. The system should be readily available for people needing help.

For the security staff’s interface it should basically show the most important information that they are currently looking for. This is the destination of the passenger. If the passenger’s destination is not on the other side of this particular security control the passenger should not be allowed through. Perhaps this could be even more simplified by just showing a red colored text if the passenger is not allowed through and a green text if the passenger should be allowed through. Perhaps this could be discussed in a dialogue with the security staff.

The system could also provide information that the security staff do not have today but might be useful to them. Like the some kind of information about how many passengers currently have checked in and might be passing through their security control. This of course has to be developed in dialogue with the staff. What should be used to display that information? A diagram perhaps?

Environment profile
According to the authors the best way to make an environment profile is to visit the place where the system is to be implemented. To us this is may seem almost a bit too obvious. Would it really be possible to make a working system without visiting the place where it is to be implemented? Sometimes maybe, but in our case no!

Aspect no. 1
Since the environment is such an important aspect of our system we need to have a good and true picture of the airport. If we do not have the correct and sufficient information how can we help to guide the passengers around the airport? If we know what the user want to find on a map, we must know how to guide them there and. This is possible only if we ourselves can find our way in the airport. We need to understand what difficulties the passengers have.

Aspect no. 2
The airport serves as a dynamic environment with changing surroundings. Sounds, noise might differ in time as well as in space. Busy times of the day will most likely render a noisier environment. At the same time there will be parts of the airport that are busier and therefore noisier than others. This is important to keep in mind while thinking about the possible sounds that should be incorporated into the interface. If there are big windows different times of the day while give changes in the light. Screens used by the passengers should be visible all the time and annoying reflections should be avoided.

To get the environmental profile we will have to conduct thorough investigations around the airport to get a concept of how it looks. Perhaps this will be done together with a domain expert. While walking through the airport the domain expert should point out areas that might need extra attention.
Profiles put together
All of the profiles above will be documented by text, photos, drawings, statistics and anything else that will contribute to the understanding of them. This will be done continuously throughout the process. Reviewing these with the team members gathered will hopefully get us a clear picture of what the system should contain. Still this is not a single step in the process, but something that should be done throughout the whole process of developing the system.

Task modeling and Use cases
There are many ways to model work. Most traditional ways to describe work or tasks are variants of the kinds of models that should be quite familiar to software developers. One popular approach is to use some form of flowcharts for e.g. finite state machines, workflow models, data flow diagram etc. for describing tasks in terms of logical sequences of events or processes. Our job in this project is to (like we mention earlier) create an open source invention environment for the national airport and for us to do that we need to have a clear view of how the system and user interaction will look like. To do that we going to use task modeling and it will look like following:

We going to describe each user (in its own model) based on the requirements and after that merge every model together, so we can see exactly how the different user models depend on each other.

According to the requirements we have 3 different of users, the passengers, the security staff and the terminal staff.

Passenger:

![Figure 1. Passenger GUI](image)

Security staff:

![Figure 2. Security GUI](image)
Terminal Staff:

![MultiTouch screen](image)

Passenger flow

Figure 3. Terminal staff GUI

It’s important to understand that these models above are just a brief explanation how the system should work and not an exact description of the GUI. The exact description model of the system using task modeling can be done in many different ways, and the complexity level is very high so we are just going to show how a real task model can look like. Here we are going to use one of the most common methods by using CTTE (ConcurTaskTreesEnvironment) tool. But before showing that we will explain what CTTE is and how you use it.

CTTE - Is a graphical notation method for task representation. CTTE enables to create task trees and to specify task properties. The basic task categories in CTTE can look like following:

<table>
<thead>
<tr>
<th>Abstract</th>
<th>Interaction</th>
<th>Application</th>
<th>User</th>
<th>Cooperative</th>
</tr>
</thead>
</table>

We are going to show a small example how it can look like by using one our tasks above, which is the Flight info.

Scenarios & Use Cases
Scenarios are another way of representing the structure of tasks and work. Scenarios works in a narrative way of description, where you explain in detail what is happening. This type of representation can sometimes be inefficient from a UI design point of view. The reason is because scenarios comprise plausible combinations of individual tasks or activities and that makes it more difficult to isolate and understand the basic kernels of interaction. Because of this reason, we have decided to not use scenarios in our airport project.
Use cases is another type of scenarios but the differences between them is that the use cases describes “typical end-user scenario” and it’s a functional analysis tool which differ from scenarios. For this project we need a functional analysis tool which can help the user to understand the system in a good way, so we are going to choose use case modeling for that. In this report we are going to demonstrate with an example of one of our functionalities how a typical Use Case could look like:

Example: To find out where your luggage are going?

1 step: Find one of our Information Machines.

2 step: Scan your passport.

3 step: Choice the luggage information option.

4 step: Check if your luggage id and Passport Number match with each other.

5 step: Check the destination of your luggage.

Help system and documentation

The development of help system and documentation runs through the whole development process. The help provided by the system can have a profound effect on the usability of a system. The good help system which supplies well-written, well-organized, and accessible help can answer some questions in the use of system, such as what is this? How do I do that? Where do I find it?

Our system can offer tutorials and encouraging support for new users, in our case that would be the passengers most of the time, it will also cover help for the more advance options that the security and terminal staff (and even experienced flyers) might want to learn and use. This will promote a continuous learning among improving intermediates.

However, we have to keep in mind that the average developer is not the expert on writing good documentation. So in our project, we need the specialists to write the help documentation and files for easiness of use of our passengers, security staff and terminal staff.

To organize the help system, we will use help cases. Use cases are natural for organizing and providing access to help because they represent the basic interests of users. Each essential use case is a complete and well-defined task based on something a user might try to accomplish. The well constructed essential task model will reflect how users think about and conduct their work. Each use case becomes an entry in the help file. (Software for use, P236)

Usability Inspections

When looking to improve the design of our user interface the most important thing is to get to know where we went wrong, that’s why it is so important for Larry Constantine and Lucy Lockwood to get Usability Inspections in their abstract software model. The first one once the
Implementation Model and Operational Contextualization are done and the second one at the end of the method to review if the product obtained is what we were looking for.

We have to keep in mind that usability inspections do not replace fine development practices and design.

For our new Airport Terminals we would contract usability experts to evaluate our software with a special focus on problems that could confuse the passengers using it, users that would not use the system really often and as such cannot be bothered with complex interfaces.

Experts usually apply the techniques of “Studied ignorance”, take the point of view of a completely naïve user, “Over the edge”, pushing the interface to its limits, trying to think out of the box on what -weird- ways the interface might be used, and “Every mountain”, carefully checking every control, tool and command button in the interface.

Experts are a cost effective option that is capable of detecting 60-70% of most usability and main problems in a really short period of time, a good option to start with and then keep polishing the minor problems.

User reviews would be used, in collaborative usability inspections, at the last Inspection phase to identify further problems that have passed by the previous inspections controls. More on this later.

Peer reviews is not an effective and productive way of finding usability problems.

**Inspection Method**

The chosen inspection method to conduct the inspection would be the Heuristic Evaluation. This method, created by Jakob Nielsen of Bell Labs and SunSoft, has defined a set of general rules or heuristics for good user interface design, a total of 10, that for our terminal software we will use [Chapter 3 of Constantine and Lockwood book] though other sets of heuristics can be applied. With this method we would have different evaluators that have been trained in this method, all of them would independently inspect our design in a search for usability problems that are related with the defined heuristics. Once done their results are pooled.

Since the scope of our project is not small each inspector would be assisted with a member of the development team, as an observer, this would allow the inspector to make questions about the application domain, the software itself and it could also serve as a source for hints or suggestions. The developer would also allow for a more efficient use of the inspector’s time by taking himself the needed annotations about the inspector concerns.

The inspection is done in a 2 pass plan. The first one is a general one where the flow of the interface is inspected, in the second one each component and its content is inspected according to the standard 10 heuristic rules we are using from Nielsen. Each of these inspectors would identify around 35% by themselves, with results pooled and knowing than on average 2 inspectors will find around 50% of the problems and 5 a 75%, considering the size of
our project having 3 inspectors would be the chosen option for us to achieve a good cost-effective solution.

**Collaborative Usability Inspections**

Collaborative usability inspections would be conducted at the last phase of the 2 of the Usability Inspections present in our process. A collaborative usability inspection is formed by a group of software developers, end users, experts and usability specialist that perform a deep inspection of the obtained final product. This will be done when the final product obtained has passed successfully the previous Usability Inspection and has corrected the problems encountered by the inspectors in that time. The end users present in this group to achieve our goals would be mainly profiles of airport passengers (young and old, more used to travel by plane and less...), and then a couple persons from each of the staff groups at the airports (Security, Terminal workers, Airline workers).

During the collaborative usability inspection is essential to have a proper attitude, focused on each detail of the interface and always from an impatient and intolerant user point view. This cannot be mistaken with how to criticize our Airport Software, the criticism needs to be directed toward the product and never to the people who developed it or designed it.

The team would have some defined roles, the Lead Reviewer who would be leading the whole inspection, the Inspection Recorder that keeps record of all the defects found with complete information about them, the Continuity Reviewer who checks the flow of the whole interface through its different screens, (junior) Developers who can give a fresh point of view on the usability software, Usability Specialists, Users and domain experts to give a more real point of view.

The main objective of our collaborative usability inspection is to find usability problems, not solving them, at the same time the effective usability measures are also identified but as a second objective. Finding this defects would not be considered a failure, quiet the opposite, the objective of these inspections is to search for as many as these problems as possible, which would mean a success on the process.

**Inspection Process**

Our inspection process would be the one based on the book from Constantine and Lockwood.

It would consist of a Preparation phase where the Lead Reviewer would make sure that all needed logistic and equipment is in place. In this phase it is better if none of the other participants review the software prior its inspection, it is better to get everyone started from a fresh start and then advance together in the inspection.

The Preparation phase would be followed by the Interactive inspection, with the software system in the beginning state, from there the team would start executing typical, critical and representative tasks with the system, these user cases should be prepared in advance. It is important to give users their space to try out these user cases but always careful to not give them too much time and attention or it might become a user review with a lot of non-user persons sitting around wasting the time.
On each interaction context that is executed for the first time the person assigned as Lead Reviewer need to ask for the first impressions of the rest of the inspection team, keeping that attitude till all screens in the flow of each case scenario is checked.

The Static inspection phase follows up, its focus is on checking each element and screen from our Airport Terminal software, it doesn’t have to follow a flow as in the Interactive inspection but focus on the details of every component independently.

Finalization and follow-up of the inspection, once the inspection is completed it is important to save some of its time to do a fast review of all usability problems recorded. Then it is time to analyze the impact of the errors found and plan ahead as explained in Constantine and Lockwood book [Chapter 16]:

- Estimate the cost and difficulty to repair them.
- Organize them by functional category, severity and cost of corrections for defects.
- Check and review the user interface architecture for possible reorganization or refinement.
- Grouping defects. Then schedule its fixing and assign them to particular groups of developers.
- Completing redesign of affected portions of the Airport Terminal interface.
- Get the changes implemented.

Detecting defects doesn’t mean that all of them will get fixed (depends on how important it is, how much time it would require to fix, if it affects the flow of the user interface...), but it is better to know it is there and stay in the darkness.

**Conformance Inspection**

We consider this last part of our inspection process quiet important to introduce, it should be based on a group of 3 participants that are familiar with style guidelines and user interface standards. Their focus will be to identify that the new developed software doesn’t leave too much the standards and conventions of user interface. This allow, specially to new users, that when interacting with our new software they can make more “intelligent guesses” when in doubt on how to achieve their goal.

Following this inspection model based on what has being learned from the book “Software for Use” by Constantine & Lockwood should give a really well depurated software product where most or almost all of its usability problems are resolved in a reasonable time, with many points of view taken in mind when looking for problems on the software and also when looking for answers on how to fix them and with a schedule that can be predicted with a good amount of accuracy.
### Project Timeline

<table>
<thead>
<tr>
<th>Activities</th>
<th>Participants</th>
<th>Number of work hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>(*) Collaborative Requirements Dialog</td>
<td>Users, Software manager, system engineers, human analysts</td>
<td>8h/person<em>3persons</em>5days =120 hours</td>
</tr>
<tr>
<td>(*) Domain Modeling &amp; Operational Contextualization</td>
<td>Users, Software architectures, Software Designers</td>
<td>8h/person<em>2persons</em>10days =160hours</td>
</tr>
<tr>
<td>(*) Task Modeling &amp; Interface Content Modeling &amp; Implementation Modeling</td>
<td>Users, Software engineers</td>
<td>8h/person<em>3persons</em>14days =336hours</td>
</tr>
<tr>
<td>(*) Usability Inspection</td>
<td>Users Human &amp; Computer interaction engineers</td>
<td>8h/person<em>2 persons</em>6days =96hours</td>
</tr>
<tr>
<td>Object Structure Design &amp; Concentric Construction &amp; Architectural Iteration</td>
<td>System engineers, Software designers</td>
<td>8h/person<em>2persons</em>20days=320hours</td>
</tr>
<tr>
<td>Help System and documentation &amp; (*) Standards and Style Definition</td>
<td>Users, Documentation Specialist, Software engineers</td>
<td>6h/person<em>2persons</em>4days=48hours</td>
</tr>
</tbody>
</table>

Users (passengers, security staff, and terminal staff) are involved in the activities which are marked by *.
Advantages & Drawbacks
The following points have been found as interesting advantages and possible drawbacks when implementing the software process suggested by Constantine and Lockwood in our project:

+ → Concurrent process system that allows a lot of flexibility on when to develop each activity.

+ → The process allows giving more importance to some activities than others, even skipping them if the project doesn’t require them (Except for the core activities: Domain Modeling, Collaborative Requirements Dialog and Task Modeling).

+ → Abstract models encourage innovation. It allows for more possible options since it gets the developers to fill the gaps in more imaginative ways.

+ → One complete Usability Inspection right before the implementation can help to reduce a lot possible errors that would be entered in the system, leading to a less effective use of the developing time.

+ → This user-centered approach gets the user very well involved in the key activities through the whole process.

→ → Though the inspections are really complete they need to involve a lot of different people from different areas and even different companies, making it possibly hard to achieve, especially in really big projects. It’s a win-loose situation, if it cannot be properly scheduled for everyone in a short period of time then we might lose some important points of view, otherwise it might need to be schedule late in time leading to delays in the planning, needs to be considered in a per case basis.

+ → Good and complete view of the whole project through the Task Models & Use Cases activities.

+ → The Operational Contextualization idea of being developed and extended throughout most of the process allows the building of a more proper interface in consonance with the environment and domain limitations.

→ → Big and steep learning curve to develop the whole Task Modeling activity at once and in a proper and clear way, even harder if it wants to be done at the same time with the Interface Content Modeling activity as the book suggest as a possibility. A solution would be the segmentation of the Task Modeling for the tasks that can be achieved and then once done, stick them together and revise the whole view with everything. Also having a previous phase where a prototype or various prototypes are done for a better explanation of the tasks to Task Model.