**“User-Centered Process for Designing and Implementing Interactive Motion Tracking Software”**

1 Introduction

When designing a completely new software system there are many factors to consider but the most important ones are the future end users. This is especially true for systems aimed a narrower group of end users as it will provide the possibility to tailor make the software to a much greater extent. Depending on the background of the end user restrictions can be put on the software system and demands might vary greatly. As a result of this involving the user in the actual software development process can produce a superior software system, the option to involve the user in the entire process might not always be available but if it is the user can prove to be a invaluable asset in the software development process.

Motion vision is very important for any seeing species in the animal kingdom, for example how animals track moving targets in their field of vision or how they generate an optic flow as they themselves move to be able to go in the desired direction (Borst and Euler 2011). There are several methods for studying motion vision, one important one is to look at behavior as a function of visual stimuli. Using insects for doing behavioral analysis is a common technique as studying vertebrates is a complex endeavor and surprisingly enough certain insects code motion vision in a fairly similar manner as vertebrates do.

Getting a better understanding of motion vision is important for advanced technology such as self driving cars, air planes, robots and just about any moving artificial intelligence.

2 Background

* 1. Motion vision at Uppsala University

At the motion vision lab in the neuroscience department of Uppsala University fly's, and in particular hoverfly's, are used for their research. For doing behavioral research different methods are used, for example you can record the fly's movement in a contained area while presenting visual stimuli for it and then do image analysis of the recorded material to get the necessary data. Another technique that the lab at Uppsala University is using is a trackball setup on which a tethered fly can walk on to generate a virtual movement path.

The setup is built using two optical sensors extracted from two high quality gaming mice. These are held in place in such a way that they are perpendicular to each other and aimed at the center of the cup where the ball is placed. Using light airflow streaming from the bottom of the cup the ball is hovering slightly and very easy to spin which is necessary for the fly to be able to rotate the ball. The fly itself is then tethered to a tube so that it hangs low enough to be able to walk on the ball and high enough not to be pushed into it. Furthermore the fly is aimed at a CRT monitor which in the experiment is going to be used for showing visual stimuli.  
  
The mouse sensors on the trackball setup are going to be used as simple motion detectors. This means that there is a need for specific software that can read the raw mouse data and decode this to the fly’s movements. Using mice like this is not common and hence there is hardly any ready-made, available software for this purpose.

* 1. User-centered development

3 Purpose and methodology

* 1. Choice of development process

As I as a developer and designer worked in the very same lab as the end users in the motion vision lab at Uppsala University choosing an agile development process was natural. The initial phase of the project, also called phase 0, was devoted to project description. This includes finding and analyzing requirements of the software system, design a suitable graphical user interface and planning the implementation phase.

The implementation phase was divided into four sprints of two weeks each. Each sprint was finalized by having a walkthrough of the progress so far where FlyTracker was tested by checking of the list of requirement for that particular sprint. This made it possible to every second week get very valuable feedback and to make sure that the development was on the right track. Furthermore, this also made it important to divide the work such that functions (or requirements) were completely implemented during its respective sprint. In the middle of every sprint there was also a shorter check-up meeting where we discussed potential problems or changes to the requirements before they were fully implemented.

Working according to the principles of user centered design was also a clear choice

* 1. Working in sprints

Agile methods are iterative and the work is done in sprints. The choice of agile methods defines exactly how the sprints will look but there are features which are applicable for most agile development processes. For my project some of these features weren't applicable as for example I have been operating both as the usability team and as the development team. This means that any communication normally needed between these groups is void for this specific project.  
  
Agile user-centered methods requires continuous feedback from user and as I have been working closely with the lab group that is going to be the end-users daily feedback has not been a problem. For broader planning and discussions of larger issues official meetings were held. Finally, at the end of each sprint I performed a show and tell of the work I had done so far to check of user stories and requirements that was implemented during the current sprint.

* 1. Evaluation methods

4 Theoretical framework

* 1. S

Usability is a very important term in human-computer interaction (HCI) and is a measurement you can use in an evaluation of a systems user friendliness. What is meant by usability is that usage of the system should require as little effort as necessary when performing the tasks needed. The system should provide the user with necessary information and that it is organized in a natural way in the graphical user interface. That the software has a short learning curve is also a feature of a system with high usability.[[1]](#footnote-1)

Design principles are an important tool for shortening the learning curve of the system for naive users. Design principles are artifacts that are often found in similar systems that are easy to recognize for the users. This means simple artifacts such as undo-/redo buttons, back buttons as a way of navigating and generally just placing interface items where they normally would be in a similar system.[[2]](#footnote-2)

Emphasis on usability tends to diminish in the software process for two reasons in particular. One reason is due to the fact that the customer rarely specify the fact that they want a usable software system (they think it's implied) and when the company delivers the software it is not as user friendly as the customer might have wanted. This is because the customer probably isn’t a HCI expert and might not even know about the term usability although on a abstract level they want a system that is easy to use but never realize they need to emphasize this. Therefore this is something that is important for the designers to bring up and discuss with their clients. Another issue that might arise is that usability experts are normally only actively participating in the early parts of the software development process and not as much in the implementation and testing phase. This is mainly because usability experts rarely are computer scientists. What is needed to solve this is HCI experts that are familiar enough with programming to be able to participate actively in the implementation phase in particular but also in the validation phase where system requirements are validated (including usability requirements).[[3]](#footnote-3)

* 1. Why agile user-centered methods?

Agile methods are the name for the group of software development processes where the project is divided into shorter iterations or sprints. Each iteration may last a few weeks where in the beginning goals for that iteration are defined and in the end the result is presented for the managers. During the sprints most of the phases of the software process are gone through, requirements are analyzed, designed for and implemented. This way of working has several advantages, first of all, in a rapidly changing environment that the software system is created new requirements may appear and old ones might be in the need of change. Following a plan based approach; months of work might be lost because of the huge amount of initial planning.

The key is in other words rapid and dynamic development which is also in line with what suitable for processes where the end users are involved, user-centered development processes.

In plan-based development a lot of focus early on is put into documentation, in agile processes however documentation as a self-serving purpose is not done. More focus is instead on working close to the customer or stakeholder to be able to quickly communicate face-to-face and discuss issues or changes.

Continuous feedback from the users is another important aspect of agile methods, its not always possible to get continuous feedback during the sprints but instead a feedback session can be held at the end of each sprint. During which the developer team show the newly implemented features and the stakeholders can give feedback. Barring any major misunderstandings or changes in directions from the stakeholders this session is mostly about updating where the developers stand and provide closure for that particular sprint.

* 1. Phase 0

The initial phase of a user-centered systems design is called phase-0 and contains the overall project description.

**4.3.1 Contextual inquiry**

Contextual inquiry means that the team is performing an analysis of the end-user by observing them in their work and seeing how they perform tasks as they work is what is known as a contextual interview. In this phase design of the system is not focused on but rather how the working environment looks like and how work generally is done in the particular context. Focus is not so much as looking for problems but rather to understand the user(s) the system will be aimed at.*[[4]](#footnote-4)*After the contextual inquiry all notes and data gathered about the user needs to be organized in a logical fashion. The data might be overwhelming though and it can be difficult to structure, therefore an *interpretation session* is held. *Affinity diagrams* are a way of organzing the data. These diagrams are built bottom up by building small groups of items that have a connection and then iteratively build larger groups of the newly created groups.[[5]](#footnote-5)

**4.3.2 Sequence models**

*Sequences models* are models describing strategies for performing specific tasks and define the intent and the following steps that are necessary to perform to finish the task. The model describes in detail exactly what the user do, this is to get a good view of what is actually happening and where issues might lie. The sequence model is a good tool to possibly find better strategies than the ones currently employed by the users.[[6]](#footnote-6)

**4.3.3 Personas**

**4.3.4 Scenarios**

**4.3.5 Prototypes**

Prototypes are also a good way to early in the design phase provide a clear way for the user how the interface is supposed to look like. There are ways to let the user try these out whether the designer is using low-tech prototypes such as paper prototypes or via some prototyping software creating a interactive interface. If the case is the former the designer can use post-it notes as pop-ups and covering invisible parts with paper and removing it as the user makes them visible by its interactions with. Paper prototypes are a good tool for getting the user to get a sense of the look and feel of the future system without actually having to program a GUI. This makes it easier to do major changes to the structure of the GUI as you wont have to program it. As the project moves on these prototypes won't be sufficient but as the actual GUI is produced the user can give feedback on that instead and hopefully any future changes will not have to be major.[[7]](#footnote-7)

* 1. Release planning phase and sprints

In the release planning phase the sprint planning is done, first of user stories are created on so called story cards. These describes requirements for the systems from the user's perspective. The description consists of what the user want to do and for the development team this description is all that matters and any other restrictions or demands on the task needs to be specified in their own user story. In other words stories will be short and this serves another purpose in that they can be implemented quickly which is a great advantage in the agile process. Developers can implement a new story within hours or days and if there are problems which renders the story obsolete not much time has been wasted.

After all story cards are created the sprint planning can start, usually a sprint will go on for a few weeks and this includes both implementing and testing. Each sprint will be allocated a certain amount of stories where the number depends of the complexity of the stories. The planning should makes sense in that any stories that depends on other stories need to be scheduled after their dependencies. Furthermore, stories that are more important should be implemented early on and so should stories that are dependent on complex technology. In the case of issues with that particular technology there is than still time to fix these compared to if those stories would have been pushed back to end of the development phase.[[8]](#footnote-8)

* 1. Usability in the software process

Performing a user-centered system design as a part of the overall software process means that you will need to implement a user-centered requirement framework in the selected software process. Zimmerman and Grötzbach suggest one framework where they introduce three types of non-functional requirements, usability requirements, work flow requirements and user interface requirements.

Usability requirements are..

Work flow requirements are a description of how the software system is supposed to support the user when trying to perform certain tasks. They are a description of what actions the user will need to take and how the system will achieve these. The actions needed to be performed can be described in analysis artifacts such as use cases or scenarios. These requirements can when the system is finished be used to evaluate the usability of the system and the requirement itself can be validated and see that the actual work flow corresponds to the requirement.

User interface requirements define how the interface should look like and translate into design artifacts such as sketches, navigation models, information architecture and eventually paper prototypes. [[9]](#footnote-9)

* 1. Methods for evaulating usability

Evaluation techniques are often grouped into two categories, namely expert evaluation and participant-based evaluation. The latter means that end users or a group representing them will be a part of the evaluation, mostly by actually using the system and having to answer questions or surveys. Expert evaluation means that the system under design is evaluated by usability experts. This method however should never be used by the designers themselves as they could have significant bias towards the system as they know it very well and could potentially both find too few problems or obscure problems that aren’t realistically going to occur during regular use. [[10]](#footnote-10)

One example of participant-based evaluation is cooperative evaluation. This means that a user will try performing predefined tasks (these tasks should of course be part of realistic future use) together with the expert performing the evaluation. All this could be video- or audio recorded to get the most out if it but it can also be sufficient that the expert is taking notes on how the program is performing. During the process the participant will be encouraged to talk out loud and the expert will also be asking a series of questions. For a detailed description of guidelines for a cooperative evaluation see table 1.[[11]](#footnote-11)

1. Table - Guidelines for cooperative evaluation. [[12]](#footnote-12)

|  |  |
| --- | --- |
| Step | Notes |
| Using scenarios prepared earlier, write a draft list of tasks. | Tasks must be realistic, do-able with the software and explore the system thoroughly. |
| Try out the tasks and estimate how long they will take a participant to complete | Allow 50 percent longer than the total task time for each test session |
| Prepare a task sheet for the participants | Be specific and explain the tasks so that anyone can understand |
| Get ready for the test session. | Have the prototype ready in a suitable environment with a list of prompt questions, notebook and pens ready. A video or audio recorder would be very useful here. |
| Tell the participants that it is the system that is under test, not them; explain and introduce the tasks | Participants should work individually – you will not be able to monitor more than one participant at once. Start recording if equipment is available. |
| Participants start the tasks. Have them give you running commentary on what they are doing, why they are doing it and difficulties or uncertainties they encounter. | Take notes of where participants find problems or do something unexpected, and their comments. Do this even if you are recording the session. You may need to help if participants are stuck or have them move to the next task. |
| Encourage participants to keep talking. | Some useful prompt questions are provided below. |
| When the participants have finished, interview them briefly about the usability of the prototype and the session itself. | Some useful questions are provided below. If you have a large number of participants, a simple questionnaire may be helpful. |
| Write up your notes as soon as possible and incorporate into a usability report. |  |

Heuristic evaluation is a form of expert evaluation where the expert evaluates the system in regards to usability benchmarks (heuristics). Jakob Nielsen defines ten important heuristics and they are: [[13]](#footnote-13)

**4.6.1 Visibility of system status**

The idea here is that the system should always provide clear information to the user of what is going on so the user doesn't have to guess.

**4.6.2 Match between system and the real world**

Too much technical terms should be avoided, rather should the system speak the same language as its users so for systems aimed towards users with the same background technical terms that particular group is of course okay.

**4.6.3 User control and freedom**

Navigating back and forth should be easy and always possible, a home button for example that always take you back to the main window and cancel buttons to make sure that you can abort when accidentally pressing a button you didn't mean to press.

**4.6.4 Consistency and standards**

Follow standards for interface elements, naming of buttons and structure of interface.

**4.6.5 Error prevention**

Try reducing the amount of errors possible to make as much as possible. For errors that would cause the system to fail error prevention should be done in the code while errors that might cause the user to lose or corrupt data or settings warnings should appear prompting the user to confirm their attempted action.

**4.6.6 Recognition rather than recall**

Users shouldn't have to remember more than necessary, keep as many elements visible at all times as possible.

**4.6.7 Flexibility and efficiency of use**

Naive users and experienced users may have opposing interests as the naive ones will want to have as much info as they need while the experienced ones will want o

**4.6.8 Aesthetic and minimalist design**

Keep information that is necessary for the user, irrelevant information will just take focus from relevant info.

**4.6.9 Help users recognize, diagnose and recover from errors**

Any error messages should be written so the user can understand them, not any technical messages. Also a suggested solution should be provided.

**4.6.10 Help and documentation**

The less documentation that is needed the better but for complex systems it might be necessary and should then exist. The documentation should be easy to search and provide the necessary help for the user.

5 Analysis and design

* 1. Contextual inquiry

As I have been working closely with the end-users in the very lab they are going to be working with the project I felt no need for a formal contextual interview but rather I got the chance to observe how work was performed all the time. I could ask questions about routines, issues that might arise and generally observing work as it happened.

* 1. Personas

The system in question is aimed at a very specified group of people and therefore design to optimize for a wide variety of users is not necessary. The end-users that has been identified during the initial phase of the project are all categorized as high level biology students described in the persona below.

The user is a 28 year old PhD student from Germany studying neuroscience. X has 2 years left before his disputation and having being a student for many years he has above average computer knowledge but not necessarily any programming knowledge. As he is a student from Germany now living in Sweden his English is also above average but also might not be perfect.

Being young and in generally good health X has no issues learning new systems as long as they are reasonably complex.

X performs extensive research in his quest to get his PhD so many hours are spent doing experiments and he doesn't like when unnecessary work has to be done to perform these experiments nor like to wait on any non-responding or unreliable systems.

* 1. Scenarios

X has finally set up the experimental environment and having software for visiual stimuli installed on one computer and the data acquisition software on another machine. He now feels ready to finally start the experiments he needs for his research. Though he is comfortable with the stimuli software he has never used the DAQ-software and starts it up to look around.

* 1. Sequence models
  2. Paper prototype

The paper prototype was made in two iterations before the actual draft of a GUI was first implemented. Using a combination of the open source tool Pencil and actual drawings a prototype was produced with no interaction for the user but clearly specified what actions are possible and what they lead to.

6 Implementation (use help manual for this section)

e.

* 1. The Matlab package

The Matlab package provides the main application and in return is just calling the python files needed to read mouse data.

* 1. The Python package

DAQ.py contains the vital classes for reading mouse data.

7 Evaluation

* 1. Why cooperative evaluation?
  2. Process

The candidates for the evaluation were chosen from the institution of neuroscience at Uppsala University. Although they do not work with motion vision they are PhD-students in the same field as the potential end-users. Four people were chosen for a qualitative study of the usability of FlyTracker.

Each evaluation started with an introduction of what kind of research is done at the motion vision lab so that they get a general understanding on what the purpose is and what tasks they actually are going to perform. They were also encouraged to ask questions in case there was anything that was not clear.

There were two main tasks for the experimental subjects to perform. First of the system was to be calibrated following the steps of the calibration process but not setting up the actual calibration set up as that would be too time consuming and would not fulfill any purpose. The second step would be with the pseudo-calibrated system performs an experiment using the lab's visual stimuli software and moving the ball manually to simulate fly movement.

* 1. Result

8 Results

9 Analysis and discussion

10 Conclusion

Bibliography

Articles:

Borst, A and Euler, T “*Seeing Things in Motion: Models, Circuits, and Mechanisms”*, Neuron 71, September 2011

Moore, Richad, Taylor, Gavin, Paulk, Angelique, Pearson, Thomas, van Swinderen, Bruno and Srinivasan Mandyam, ”*FicTrac: A visual method for tracking spherical motion and generating fictive animal paths”, Journal of Neuroscience Methods, 14*

1. Benyon, David, *”Designing Interactive Systems”,* s. 84 [↑](#footnote-ref-1)
2. Benyon, D s.89 [↑](#footnote-ref-2)
3. Göransson, B, Gulliksen, J, Boivie, I, *”The Usability Design Process – Integrating User-Centered Systems Design in the Software Development Process”* [↑](#footnote-ref-3)
4. Beyer, Hugh, *User-Centered Agile Methods* [↑](#footnote-ref-4)
5. Beyer, Hugh, *User-Centered Agile Methods* [↑](#footnote-ref-5)
6. Beyer, Hugh, *User-Centered Agile Methods* [↑](#footnote-ref-6)
7. Benyon, s.184-187 [↑](#footnote-ref-7)
8. Beyer [↑](#footnote-ref-8)
9. Zimmerman, D, Grötzbach, L *”A Requirement Engineering Approach to User Centered Design”* [↑](#footnote-ref-9)
10. Benyon, s.228-232 [↑](#footnote-ref-10)
11. Benyon, s.232 [↑](#footnote-ref-11)
12. Benyon, s.232-233 [↑](#footnote-ref-12)
13. 140207, http://www.nngroup.com/articles/ten-usability-heuristics/ [↑](#footnote-ref-13)