Abstract

The project is about understanding how user behaviour affects power consumption. There is uncertainty when it comes to the relation between power consumption and user behaviour. This project aims to bring a better understanding of user behaviour and in what ways PC applications affect power consumption.

The project is structured in two main components, software development and data analysis. The first is the development of a software application which gathers data about user behaviour in terms of Windows processes. This tracking application is run during user trials and gathers entries about the running processes into a database, as well as changes between different power states. The tasks that a user undertakes are therefore symbolised by the processes that are running the applications opened by the user. Task duration, task focus, user activity are just the few of the information data gathered for the purpose of the data analysis. The tracking application is built to carry “in-the-wild” monitoring, meaning that it does not affect or modify user behaviour, therefore not affecting the analysis of the gathered data. The objective of the tracking application is to gather continuous data and to record data in terms of user activity and inactivity.

The second component was conducting the user trials and experiments in order to conduct the final analysis. The data gathered from the user trials was used to understand user behaviour and to find various patterns. User profiles and statistics were created based on this data and analysed in order to generate usage patterns. These patterns were then used to link power consumption to process usage. This was achieved by using a device that measures power consumption. Using this device, experiments were undertaken using the patterns obtained from the user trials data analysis and applying them to obtain data about power consumption. Furthermore, information was gathered from the users in terms of power management, and ratings were given to the tested patterns using information from an online resource.

- This project produced a new Windows application, which was used to track and gather data about the processes running on Windows machines.
- The data gathered during the user trials was used in analysis and experiments in order to answer research questions leading to the following discoveries:
    - Better understanding of user behaviour
    - Creating user profiles based on activities and statistics
    - Creating patterns of user behaviour for use in power consumption experiments
    - Allocating power consumption to processes and activities
    - Creating ratings based on the conducted experiments
    - Reducing the uncertainty on how user behaviour affects power consumption

Acknowledgements

I would like to thank my supervisor, Chris Preist, for his indispensable advice and guidance throughout this project.

I would also like to thank Daniel Schien for his support and for his insight on the software development and data analysis.

Finally I would like to thank my parents for believing in me and for loving me unconditionally.

Contents table

1 Aims and objectives

The project has two main aims. The first one is to design and develop a software application that can gather data about user behaviour on Windows machines in terms of process usage and power state changes. The application needs to be developed for its use in user trials in order to gather data for the data analysis. The second aim is to perform data analysis and to carry out power consumption experiments based on the analysis.

1.1 Tracking application objectives

- Communicate with Windows API and store data in a SQLite database
- Continuously gather data about the running processes on the system and successfully update the database
- Track user activity and record data based on window focus and user input
- Gather data about power state changes
- Run in the background, without requiring user intervention, without “upsetting” the user and providing easy access if needed

1.2 Data analysis objectives

- Create a set of valid assumptions about user behaviour
- Apply assumptions to the gathered data in order to create statistics and comparisons
- Create user profiles based on data and assumptions
- Create usage patterns for use in power consumption experiments
- Gather statistics about general power consumption on the computers involved in the user trials
- Run power consumption experiments to allocate power consumption to PC applications
- Create ratings based on the conducted experiments
- Create a better understanding about user behaviour and power consumption

2 Literature review

This chapter only states the important papers from the literature research and gives a short insight on what they are about. The effects they have on the project and the decisions based on them are critically discussed in the Research Analysis chapter.

2.1 User behaviour

Within the past years, there have been various surveys and reports that analyse user behaviour and look at how computers are used. One of the most important papers on user behaviour, and the paper on which the core of the research was based is the paper entitled "Computer usage in daily life" by Thomas Beauvisage (Beauvisage, 2009). Beauvisage's paper explores the use of computers at home through the investigation of user activity.

The paper deals with computer use at home. The work done by Beauvisage is based on a large-scale recording of application data from a wide representative panel of over 1000 users during a timescale of over a year and a half. The paper presents two ways of classifying computer applications in order to describe computer use. The first one analyses computer usage temporality by reflecting a tension between synchronous and asynchronous usage profiles. The second one analyses software preferences and usage intensity, therefore distinguishing five user profiles reflecting strong routine behaviours. The paper then focuses on building user profiles based on analysis on their behaviour.

Based on Beauvisage's paper, Gabriel Barata has developed an application which tracks user behaviour in terms of accessed applications and accessed websites in order to allow users to access their computer usage history (Barata, 2012). The paper discusses an application that gathers data and presents it to the user, after which it analyses how users react to past activities and trends. It points out that there are a large number of papers dedicated to web, email and multimedia usage, yet daily usage has been given little attention. Two applications were developed for this purpose, one to record all applications and websites accessed and one to visualize the data. The visualization application analyses the gathered data and displays it for users in order to view past behaviour and to think how their behaviour can be improved.

Windows switching is tackled in many papers. It basically covers user behaviour in terms of how a user completes a task by using several applications and switching between them. Oliver Brdiczka discussed about gathering data about the documents accessed by the user and how windows switching takes place between the documents (Brdiczka, 2010). The paper tackles data privacy issues by collecting data as document identifiers and switching history between the documents, instead of document contents or window titles. The idea proposed is to group documents into groups based on the tasks undertaken by the user.

Additional to windows switching, windows management also is important for user behaviour (Tak, 2009). Tak's paper discusses a tool that visualises windowing activities in terms of switching and management. The application developed for the paper produces "maps" and statistics about windows, which shows how users access several applications at the same time in order to perform tasks. Similar to Brdiczka, it proposes grouping applications instead of documents to signify tasks, but also proposes windows "maps" to show how a user can access information from several application windows without switching between them.

There were other papers that were included in the research which had similar findings to the papers referenced above, as well as some papers which provided ideas for the development and implementation of the tracking application. One of the papers develops a tracking application which collects information on all user activity in order to improve the design of video games, in the sense that it does not only collect information on the processes that are running, but collects information on all user initiated events (Kim, 2008). Another paper built on the methods and findings of Beauvisage by analysing computer usage in order to find working patterns and working behaviours for employees at work (Saito, 2011). Similar to Kim's paper, the paper by Shen discusses a task tracker which looks at various events started by the user (Shen, 2007).

Another paper that provided important findings towards the successful completion of the project deals with "Display power management policies" (Tarzia, 2010). Tarzia's paper deals with DPM policies, specifically the "Human interface device timeout", which powers off the display after a specific amount of time in order to save energy. The paper presents an alternative policy which uses sonar sensing in order to power the display on and off, depending on the user's presence. The paper focuses on studying how these different policies affect energy savings as well as user irritation.

2.2 Power allocation

The starting point for the project is to provide the data which can be then used to allocate power consumption to processes. This means that the data analysis needs to make assumptions about the power consumption of the processes.

The starting point of the research was a paper discussing the environmental impacts of tasks performed on ICT devices according to the amount of time spent by the user and according to the amount of network bandwith they require (Teehan, 2010). However instead of analysing the impact of individual tasks, the paper analyses the impact of all the tasks at once. The paper also focuses on network-enabled devices, because networks and the datacenters that support them represent an important factor for the environmental impact. For the purposes of this project, the network analysis is not relevant.

Devices like desktop computers or laptops support multiple tasks. The impact of any given task (process) is defined to be proportional to the amount of time the user spends on performing the task. In his paper, Teehan critically discusses this impact and the ambiguity surrounding the overhead. Teehan reaches the conclusion that the best approach is to avoid including the overhead in his calculations, so the overhead is distributed amongst all the tasks. Following is the equation given by Teehan's paper on the impact share of a task *t* on device *d*.

$$task\ impact(t_i, d) = \frac{time(t_i, d)}{\sum_{k=1}^{n} time(t_k, d)}$$

Equation 1. Impact of a user's task (process or group of processes)

Equation 1 can be modified by looking at the user behaviour impact, firstly the user (positive) impact, which means the active time that the user spent on a task *t*, secondly the negative impact, which means the inactive time during which task t runs. The negative impact can be calculated as a difference between the first two equations or as a standalone equation, using the idle/inactive time.

$$user\ impact(t_i, d) = \frac{active\ time(t_i, d)}{\sum_{k=1}^{n} time(t_k, d)}$$

Equation 2. Positive impact of the user

$$negative\ impact(t_i, d) = \frac{inactive\ time(t_i, d)}{\sum_{k=1}^{n} time(t_k, d)}$$

Equation 3. Negative impact of the user

2.3 Power model

This chapter discusses a possible power model for the tracking application which was considered at the beginning of the project. Due to various reasons the power model has not been implemented and a simpler solution has been used. The chapter also provides explanation for the decision to not implement the power model.

Exact power consumption based on individual processes is difficult to achieve. One of the papers that develop a power model and apply it in order to get the power consumption of a single process is written by Da Costa (Da Costa, 2010). The paper states that it is difficult to estimate the power consumptions of individual processes running on a computer and aims to evaluate the power consumption by using an indirect methodology. The methodology proposed in the paper however uses computers which operate on a Linux operating system. During chapter 3.1.1 there will be a discussion which points out why this paper will focus on generating data for Windows operating systems only.

Other papers discussing power models involved a paper that provides power model generation for software-based power estimation in cloud platforms (Smith, 2011) and another that discusses a power model for virtual machines (Kansal, 2010).

Kansal's paper provides power models for each component of the machine that uses power. The components discussed are the CPU model, which uses a light model that looks at processor utilisation, the memory model which is however design specifically for virtual machines, the disk model which again is designed specifically for virtual machines and a model for other components. Both Da Costa and Kansal give a generalised power model which includes all of these components.

$$Power = \alpha + \beta_1 \cdot CPU + \beta_2 \cdot Memory + \beta_3 \cdot Disk + \beta_4 \cdot Network\ component$$

Equation 4. Power model proposed by Da Costa

In the previous equation, the alpha and betas are unknown variables which depend on each device. Therefore creating a tracking application with default settings would not prove useful for using in user trials involving different devices. Also the proposed power model does not consider how single process power consumption relates to the total power consumption. Finally, a group of second year students from the university attempted a project which involved creating a power model for single process power consumption during the course of this year. The group had good knowledge of computer architecture and of electronics and had several months to complete the project; however it was not completely successful. Therefore this project will not implement a power model. It will instead use a device that monitors and gathers data about power consumption, based on patterns derived from the completed user trials and on other experiments.

2.4 Research analysis

This chapter explains why the mentioned papers are relevant to this project and gives the decisions based on them. It contains decisions taken towards what the tracking application needs to gather and how the data analysis will be done.

There were various views on what needs to be gathered in the research papers. Since the gathered data represents the basis of the whole project, there were many aspects that needed to be considered. Firstly, the main data that the tracking application gathers is based on Beauvisage's paper. PC applications are run by processes, therefore gathering the list of running processes was the start. In their paper, Barata and Shen also gathered information about websites, specifically the names of the websites visited by the user. Shen also gathered information about the names of the documents opened by the user. Brdiczka followed the same approach; however he used identifiers for the documents in order to keep them anonymous. These approaches brought up an important aspect, data privacy. Due to data privacy concerns, Barata encountered problems when approaching users for the user trials. They were concerned that the application would breach their privacy by gathering data such as window titles and website URLs. For these reasons, the tracking application was designed to gather data about processes, specifically process names. Gathering data about window titles and website URLs would have been useful, however data privacy concerns needed to be considered.

In terms of what details of the processes are recorded, the tracking application builds on Beauvisage's paper by recording data based on all the processes, not only the focused one. Instead of only recording the process that has focus on the computer, the tracking application records all the processes that are running. It differentiates between the main process, the process that is focused, and the other processes, which can be idle or background processes. The tracking application records different data for each process, such as the duration of the process and the duration of the process in different states.

Aside from "focus", another important aspect of user behaviour that this project follows is finding out when a user goes idle. This is somewhat similar to display power management policies. The most common used display power management policy is the HID timeout as stated in the before mentioned paper by Tarzia. The goals of these policies is to turn off the computer display when the user becomes idle, and to turn it back again when the user stops being idle.

The tracking application monitors activity by the user and differentiates between "active" and "inactive", which in the end reduces the uncertainty of user behaviour impact. There are several strategies for observing and sampling behaviour (Bryman, 2011). Even though these examples were developed for real life observing (e.g. an interviewer observing an interviewee), they are relevant to this project, since the tracking application implements them. Therefore the recording of the user behaviour is based on "incidents". Incidents here refer to changing states between "active" and "inactive", such as detecting idle time and changing to "inactive" and detecting user input and changing to "active".

An aspect that was overlooked for this project was windows switching, which was discussed by Brdiczka, Tak and Shen. In order to detect windows switching, the tacking application would need to add a new entry every time "focus" is changed from one process to another. This would mean that the amount of gathered data is much larger. Because of this absence, there is need for some further assumptions in the data analysis which will be discussed later.

Based on the papers from Beauvisage and Barata, the data analysis consists of statistically analysing processes in terms of usage, activity and grouping into tasks, as well as analysing the users involved in the trials in terms of activities.

3. Methodology

This chapter will present the details about how the tracking application was designed and built, how the user trials were designed and how the power to process use linking was considered.

3.1 Software

The full code for the tracking application is available in Appendix A. The name of the tracking application is ActiveWin.

3.1.1 Programming and API

The project is centred on a tracking application that records user activity. This application was developed in the C# programming language and uses Windows API. This chapter will discuss and critically assess the choice of programming language, the API and why the project focuses on the Windows platform.

Estimates show that Windows is the most used platform by users for desktop and laptop computers. In an estimate for April 2012, it has been shown that the Windows family counts with over 80% of the operating system share (W3Schools, 2012). Windows 7 and Windows XP are the most common operating systems used, with 51.3%, respectively 27.3% of the share. Mac and Linux have considerably lower percentages, 9.3% and 4.9%. The rest of the list is completed with other Windows operating systems, such as Vista and Windows 2003. Along with these statistics which give a good reason for targeting the project at Windows platforms, other reasons include the user trial target population, which is estimated to use Windows operating systems, as well as the functionality of the Windows API, which will further be discussed in this chapter. The target population for the user trials consists of computer science students or people who are working in an IT environment, as well as students or people which are not studying o working in an IT environment. While computing students might use a different operating system, it is still statistically proven that most of them will use Windows. As for the non-computing students, Windows seems to be the preferred platform.

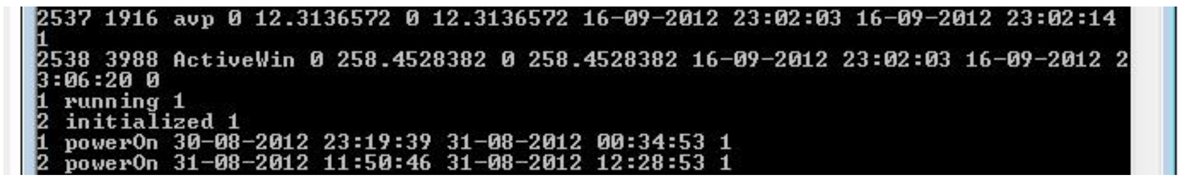
Windows APIs are dynamic-link libraries (DLLs) that are part of the operating system (MSDN, 2012). An API (Application Programming Interface) is a set of commands/functions which acts as an interface between components, in this case between the tracking application and the operating system. Using Windows APIs brings many advantages, such as accessing ready-built functions and performing direct operations inside the operating system.

There are some disadvantages too. Windows APIs represent a special category of interoperability. The functions are written in C, they do not use managed code and the data types are different from the ones used in C#. This introduces a special type of interoperability called platform invoke (PInvoke), which is a service that enables managed code to call unmanaged functions from DLLs. Although implementing the Windows APIs in C# was a demanding job, this programming language was the most suitable for this project.

3.1.2 Software design

The application was designed as a Windows Forms application, which is an event driven graphical application. This means that the user's interaction to the tracking application is through a graphic user interface. The user can interact with the application by showing/hiding and starting/stopping the application. The application registers itself in the windows registry, so that it will start on its own at every Windows start-up, without the need for the user to start it. One of the objectives of the application is to work without user interaction and without disturbing the user. Therefore after the application has been started, the user closes the tracking application window and the application keeps on running in the background. The application is displayed as an icon in the system tray, which the user can interact with.

In terms of data gathering, the tracking application stores information into a local database. The chosen SQL database engine for the project was SQLite, due to the ability to implement a self-contained, serverless, zero-configuration, transactional and embeddable database engine (SQLite, 2012). This means that there was no need for the participants in the user trials to install any package needed for the database engine. It also meant that accessing the data in the database can be done fast and reliable.



```
2537 1916 avp 0 12.3136572 0 12.3136572 16-09-2012 23:02:03 16-09-2012 23:02:14
1
2538 3988 ActiveWin 0 258.4528382 0 258.4528382 16-09-2012 23:02:03 16-09-2012 2
3:06:20 0
1 running 1
2 initialized 1
1 powerOn 30-08-2012 23:19:39 31-08-2012 00:34:53 1
2 powerOn 31-08-2012 11:50:46 31-08-2012 12:28:53 1
```

Image 1. Database entries as seen when written to console; first two entries belong to the processes table, next two belong to the settings table, last two belong to the states table

In the created database, the tracking application keeps and updates three tables. The first contains information about the running processes (Processes table), the second contains information about the power states (States table), while the third contains settings information needed by the application to run (Settings table). The Processes table contains all the information needed for the data analysis, such as process name, id, running durations (active/inactive, focused/unfocused), start time, end time and finally a "closed" value which signals that the process was closed. The States table contains entries about two power states, power on and power suspended. Power suspended symbolises either a sleep or hibernate state, while the power off state can be observed as the gaps between the other two states. Finally the Settings table only contains two entries, one for "initialisation" which is used by the application to check that the database was initialised and that the registry key was added (stops overwriting), while the "running" entry is used by the application to check if it was running the last time the system was shutdown or the last time the application was exited.

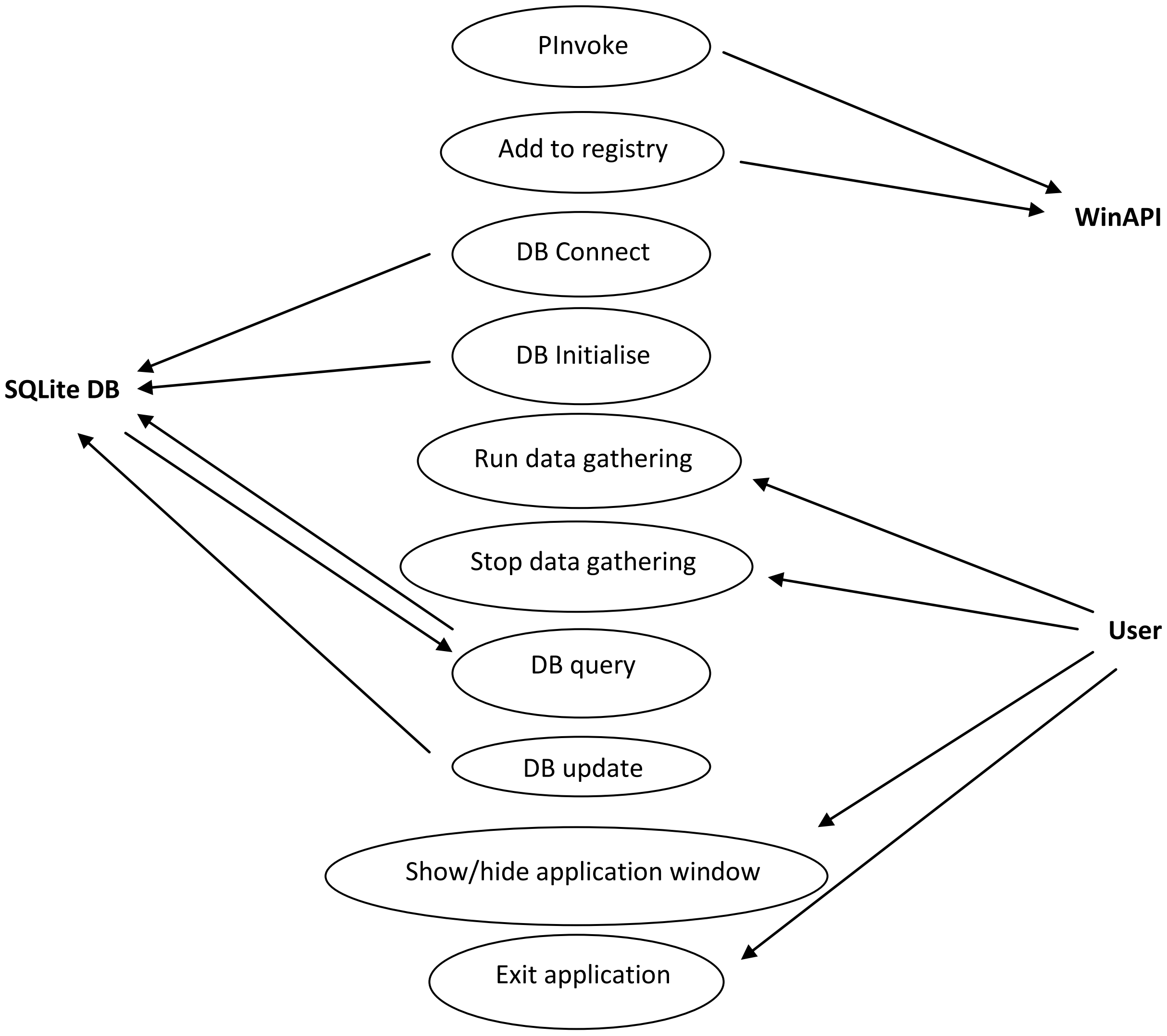


Diagram 1. Use case diagram

The use case diagram presents the main interactions between the tracking application, the user, the database and the operating system API. Firstly the tracking application creates the database, connects to the database and then initialises the database. The next step is to add a registry key entry to Windows start-up. Then all the platform invokes are setup, so that the tracking application can have access to the information it needs to gather.

When this is done, the windows form appears as a graphic user interface, giving the user the possibilities to run the application or to stop the application (this action stops the data gathering but does not exit the application). The user also has the options to hide the application (through the close button, which does not exit the application and through the system tray icon) and to show the application (through the system tray icon). When the application is running, it is updating its information on the Windows DLLs, then querying the database to compare the two sets of information, and upgrades the database where required. At the end, the user has the option to completely exit the application (from the system tray icon).

```
runProgram()
while (check program is running)
    get focused process
    get list of processes
    foreach process
        if (in database and not closed)
            if (focused process)
                if (check user input)
                    update entry as focused and active
                else
                    update entry as focused and inactive
            else
                update entry as unfocused and inactive
        else
            add new entry
```

Diagram 2. Description of the part of runProgram() which updates the Processes table

The main activity of the tracking application consists of obtaining information from the Windows API and updating the information in the database. This activity is being handled by the function runProgram(). Having created the three tables in the database, it first adds a “power on” entry to the States table, and updates the previous entry accordingly. Next it obtains the list of running processes and the focused process from the operating system. The Settings table is also updated to signal that the application is “running” (in case the system crashes or shuts down, after reboot the application will start again on its own). The next part is the bulk of the application. Using the list of processes obtained from the operating system, it compares it to the entries in the Processes table. The comparison is made between the process id, the process name and the “closed” value, which signals an already finished process. During this stage, if the last user input was made more than 60 seconds before, it updates all processes as being inactive. If not, all processes except the focused process are updated as unfocused and inactive (time is added to these two values), while the focused process is updated as focused and active. The processes in the list from the operating system which are not found in the database are then added as new entries.

Stopping the application from gathering data is handled by stopProgram(). This function sets the “closed” value for all the process entries and also sets the “running” entry in the Settings table, so that the application will not start gathering data automatically at the next boot-up.

3.1.3 Issues, problems and difficulties

Designing and coding the tracking application provided to be a challenge. There were many problems in coding and running the application.

First of all there were problems with transactions between the application and the database. The first implementation worked very slowly and was liable to system crashes (if the system crashed, the database could have been corrupted). This problem was overcome by having committed transactions. This solution also meant that the tracking application would only consume a maximum of around 20KB of memory at any given time, such that the system would not get slowed down by the application.

The application also needed to contain a configuration file in order to work on any computer. This configuration file meant that any user trying to run the application on his computer needed to install the .NET framework version 4.0. In the end this did not prove a problem, since most participants in the user trials already had the framework installed, while the others managed to install it without a problem.

During the user trials the tracking application crashed for two users. The first one was on an old PC desktop which was running Windows XP, while the second one was on a laptop which was heavily used (the user was a system administrator and there were many processes running at the same time). Both users got the error code system.invalidoperationexception. After some research, this error proved to come from accessing the currently focused process. In the code there is a small gap between accessing the currently focused process and checking it against the database entries. Although this gap in the code is very small in terms of duration (code execution time is very fast), it happened that when the user closed the currently focused process window exactly in that gap, the tracking application crashed. It did not prove to be a costly problem, since both users restarted the application successfully, and the databases were left uncorrupted.

The ideal entries for the States table were “power on”, “sleep” and “hibernation”. However it was fairly difficult to differentiate between sleep and hibernation by accessing the Windows API, therefore this implementation was abandoned. Instead, users were questioned after the user trials for some information about their computers, and one of the questions was whether the computer would go to sleep or hibernate. Some state entries also have false “end time” values, which were easily modified during the data analysis.

Another small issue involved getting the system tray icon to work properly. If the user exits the tracking application or if it crashes, the icon will still show in the system tray, which might misinform the user and make him believe that the application is still running. However if the user scrolls over the icon, it disappears, showing that the application is not active.

Also, the location of the database file gave some problems. By default the tracking application gets installed in the Program Files folder. This folder holds special restrictions, which meant that the database file cannot be saved in the installation folder, because it would have caused access issues. The solution was to save the database file in a special directory in Windows, were access rights are given to all applications. This directory was the Local User Application Data path, which usually in Windows can be found at C:/Users/USERNAME/AppData/Local.

An important issue with the application however is a missed design implementation, which caused some problems during the data analysis. The design was to have the tracking application set all the process entries as closed each time the system shuts down or goes to sleep. This was only achieved when the user pushed the “stop” button on the graphics user interface. What it means is that processes that are run by Windows (which always have the same process id) will only be entered into the database once. This issue proved helpful with background processes such as svchost.exe or crss.exe, but it affected the data analysis when it came to explorer.exe. Because the process is run by Windows, it only appeared as one continuous process from the start of the trial to the end. More details about this issue will be presented in the data analysis.

3.1.4 Software deployment

When the software was finished, it was packaged into a setup application using Microsoft Visual Studio 2010, the IDE that was used for developing the tracking application. This meant that all the files that were needed for the tracking application to work on any computer (such as the icon file, the application configuration file, the SQLite DLL file) were compiled into a Windows Installer setup file. Once the installation package was ready it was sent to the participants in the user trials. As mentioned before, some of them were required to install the .NET framework in order to run the application. This proved very easy, since the Visual Studio setup package has a built-in option that gives directions on how to install the .NET framework, which means that even inexperienced users were able to install both the framework and the application without any problems.

3.2 User trials

An important part of understanding user behaviour is to have user trials. The user trials were done over a certain set of participants for a set duration. User trials provide better understanding of the problem that the project tackles, as well as show possible improvements and give unexpected results. The user trials were important for two main reasons. The first was to produce the data which was used in the analysis, therefore creating a better understanding of user behaviour in day-to-day activities. The second was to give patterns usable in the experiments on the power measuring device.

An important aspect of the user trials was to make sure that the monitoring will be "in-the-wild". This means that users should not behave differently due to the fact that the tracking application is installed on their computer. Also the tracking application needed to run without affecting the user or without slowing down the system.

Users were considered in two main categories, "computing" users and "non-computing" users. Computing users are users who are either working or studying in and IT related field (or have great knowledge about computers). Non-computing users are therefore users who do not have great knowledge of computer usage, meaning that they use their computer for entertainment, communication, working or studying (non-IT related work or study). It was important to have users from both these categories in order to obtain different patterns and different user statistics.

Other different categories for the user trials were having users with different operating systems (Windows XP, Vista or 7) or users with different types of computing devices (laptops or PC). Unfortunately there were only two PC users and only two Windows XP users, while there were no Windows Vista users. The rest of the users had laptops and Windows 7.

At the beginning of the user trials each user was given a participation sheet with details about the trials. It is important to note that the participation in the user trials has been kept anonymous and that data privacy was not breached through the information gathered by the tracking application. The participation sheet also contained details on how to install and run the tracking application. It can be found in Appendix B.

At the end of the user trials each user received another detail sheet which asked them from information about their computer setup, which can be seen in the Table 6. The sheet also contained details about how to uninstall the application and how to remove the registry key added by the application from the start-up. It can be found in Appendix C.

As expected there were some concerns about data privacy from some participants, but nobody contacted refused the user trials due to the concerns. By showing them an example of the gathered data, the users with concerns accepted to participate in the trials.

4 Data analysis and results

The data analysis consisted of two parts. The first one regards the data analysis of the user trials, which looks at understanding user behaviour and on building interesting statistics about it. It also creates the patterns for the second part of the data analysis. The second part consists of using the patterns from the user trial data analysis and applying them to allocate power to processes. This part of the data analysis uses a power monitoring device, which combined with the process data gives some interesting statistics and results. Together the two parts of the data analysis aim to help understanding the uncertainty of how user behaviour affects power consumption.

4.1 User trials analysis

This chapter will contain statistics and data about the users. Assumptions will be made about what the data means and patterns will be generated to be used in the power allocation analysis.

The user trials were conducted with a number of 15 participants over the course of 2-3 weeks. Each user installed the tracking application and used it for different amounts of time. Users were targeted as being "computing" or "non-computing" users. More details on the users which concern power consumption will be given in the second part of the analysis. All users will be kept anonymous; therefore they will be named User 1-15. Throughout the data analysis the user numbers will remain constant (e.g. user "n" is the same person throughout the whole report).

User	Type	Device	Duration	Crashes
user 1	computing	laptop	18 days	yes
user 2	computing	laptop	14 days	no
user 3	non-computing	laptop	21 days	no
user 4	computing	laptop	12 days	no
user 5	computing	virtual machine	1 day	no
user 6	computing	laptop	18 days	no
user 7	computing	laptop	14 days	no
user 8	non-computing	pc	22 days	no
user 9	non-computing	laptop	21 days	no
user 10	computing	laptop	15 days	no
user 11	computing	laptop	20 days	no
user 12	computing	laptop	13 days	no
user 13	non-computing	pc	18 days	yes
user 14	non-computing	laptop	16 days	no
user 15	computing	laptop	18 days	no

Table 1. User trials participants details

The table gives the main details of the data gathered about each user. There were 5 non-computing users and 10 computing users. In terms of what computing device they used, most of them used a laptop, while two users who used a pc. It is also interesting to mention that the tracking application was also tested on a virtual machine running Windows XP by one of the users. This was intended just as a small test and the small amount of data gathered from that test was not used in the data analysis or in the statistics. The crashes column represents crashes in terms of the tracking application, not in terms of the operating system. The crashes for the two users were mentioned in a previous chapter about software issues.

4.2 Assumptions

A large part of the data analysis consists of assumptions since there is a large amount of data, but much information about how the processes were used. This is because gathering information about what the processes were used for (mainly websites visited and documents accessed) would have affected the data privacy of the users.

As discussed previously windows switching plays an important part in user behaviour. This means that many decisions about the user's behaviour are based on groups of processes that are running over the same time span. Groups of processes can act as a user task. The most common ones are doing research (using a web browser at the same time with a document viewer), having a webcam conversation (due to the fact that the conversation is over webcam, there is no need for the user to give mouse or keyboard input, which means the tracking application records data as inactive) or other scenarios.

In terms of gaming, assumptions can be made depending on the type of game. Games that require large amounts of resources usually run without other applications opened in the background, and periods of inactivity here can be considered as active.

In general multimedia content needs assumptions, so that the gathered data can be modified to be correct. When watching a video, the user does not interact with the computer (no input), so the process is being recorded as inactive. When listening to music the same situation can happen or the process can even run in the background, being recorded as unfocused and inactive. It is important to be able to make the difference between useful usage and unnecessary usage.

By taking each user individually and checking the same processes for all their entries (e.g. look at a music player, if there are several entries, all of them with at least some focused time, which means that the user accessed them and set them to do something), we can assume that either the process is used for a specific task by the user, or that it is not being used with a purpose.

Remote connection applications also represent a special category in the assumptions. In general remote connections are set either for data transfers or for remote access, which means the user will not leave the process running when not needed.

Installers and updaters are also considered for data changing. Installers may be allowed to work on their own for long periods of time, which means that no user input is found. Updaters usually run in the background without affecting the user.

Download clients can be considered as useful if there is obvious interaction from the user. If not they can be considered as running without purpose. The same can be applied for messaging clients. Generally the difference between active and inactive is made between the processes starting at system start-up and the processes being started by the user (it means the user runs the application with a purpose, even though the application might be recorded as inactive or unfocused for a long period of time).

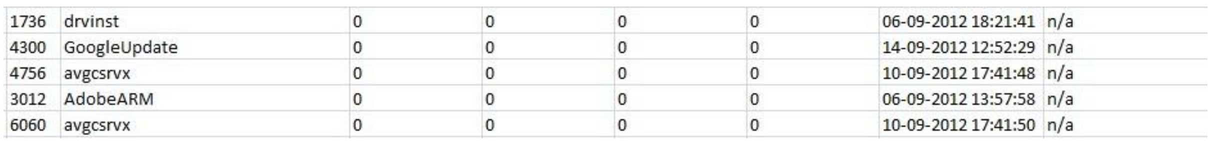
Email clients can be put in the same category as download and messaging clients. These are applications that are generally left in the background to wait for an event to happen, such as receiving a message or downloading a file. As mentioned before it depends on the interaction between the user and these applications.

There are situations in which some processes can be ignored. It happens when multiple instances of an application are open. This usually happens with Internet browsers and means that there are several tabs opened in the browser. The focus is kept by the process that opened the browser in the first place (therefore the original tab), which means that the rest of the processes belonging to the other tabs can be ignored (there is 0 focus for them).

The set of proposed assumptions are not intended as definite rules. They can either represent exactly the user's actions or they can lead to false results. However due to the fact that obtaining correct data through the tracking application is not possible, this set of assumptions needs to be applied to the data in order to produce results which are closer to reality and which tackle the uncertainty behind user behaviour. Where assumptions are closer to reality it is safe to use percentages to show statistics, where not ratings can be provided to round up the acquired percentages.

4.3 Analysis methods

In terms of raw numbers, the gathered results from the 14 users (excepting the 15th user which was a test on a virtual machine) summed up to 84027 process entries. However some of these entries were not useful for the data analysis. The way the tracking application works is that it compares the list of running processes to the list of processes in the database once every 2 seconds. Including the time it takes the code to run, the time can be averaged to 2.5 seconds. Windows has many services which only run for a brief time (close to 1 second). This means that the tracking application did not capture all of them. However these services do not count very much towards understanding user behaviour or even towards power consumption. Therefore in order to make the analysis easier, all processes which appeared as only having a start time and no end time were removed (not having an end time means that the process was only identified in the running processes list once, hence no end time was set).



1736	drvinst	0	0	0	0	06-09-2012 18:21:41	n/a
4300	GoogleUpdate	0	0	0	0	14-09-2012 12:52:29	n/a
4756	avgcsrvx	0	0	0	0	10-09-2012 17:41:48	n/a
3012	AdobeARM	0	0	0	0	06-09-2012 13:57:58	n/a
6060	avgcsrvx	0	0	0	0	10-09-2012 17:41:50	n/a

Table 2. Process entries with runtime 0

In total there were 924 unique processes that were recorded. As an interesting fact, some processes which seemed to be viruses were detected (the users which had those processes were informed and told to run a virus scan). Also because of a previously mentioned error in the code algorithm for the tracking application, some processes were saved as running for durations which spanned over several computer uses (e.g. system.exe, smss.exe, idle.exe, explorer.exe). This was due to the fact that these processes always have the same process id assigned from windows.

The recorded processes can be split into three broad categories: processes accessed by the user, processes accessed by the system which are useful for the user and processes which are accessed by the system but are not directly useful to the user. The data analysis concerns the first two categories. The processes which are not directly useful to the user are system processes or service processes which are being run by the operating system at various times in the background. The second category involves processes with which the user does not interact directly but which aid the user in various tasks. The first category is the one that sparks most interest and the one that shows most of the user's behaviour. The next table further splits the processes we are interested in into smaller categories.

Category	Description	Examples
antivirus	software that usually runs in the background to protect the computer	adaware.exe, avp.exe, avastui.exe, msascui.exe, etc.
browsers	internet browsers and additional components or plugins	firefox.exe, falshplayerplugin.exe, iexplore.exe, java.exe, etc.
download	download managers	azureus.exe, strongdc.exe, etc.
games	computer games either played locally or online	angrybrids.exe, aoeonline.exe, solitaire.exe, etc.
multimedia	software for local or online media access	itunes.exe, dropbox.exe, photoshop.exe, winamp.exe, etc.
messaging	software for online messaging and the associated trays, includes email	skype.exe, outlook.exe, yahoomessenger.exe, ymsgr_tray.exe, etc.
network	software used for network management	wireshark.exe, connectify.exe, etc.
office	software similar to the MS Office suite, editors and other management software	excel.exe, winrar.exe, isoburn.exe, etc.
programming	mostly programming software, also includes other software used for various work	devenv.exe, eclipse.exe, arcmap.exe, etc.
remote access	software used for connectivity and remote access	amazonclouddrive.exe, nxclient.exe, teamviewer.exe
system	various software used either by the system or in support to other applications, also system management software	datacardmonitor.exe, cmd.exe, sidebar.exe, taskmgr.exe, vuauclt.exe, etc.
virtual machine	virtual machines setups	vmware-tray.exe, xencenter.exe, etc.
viruses	malicious software	e.exe, freeklogger.exe

Table 3. Process classification

The table is inspired from Beauvisage's paper, where there is a similar table provided. This table however is modelled for this project. The first category is antivirus software. It is important to note that even though the processes running them will show up as unfocused and inactive in the data, they are very important in the security and successful running of a computer, therefore the power consumption originating from these processes is in good use. The browser category is very important to the analysis, since most user activity happens in terms of accessing the Internet through browsers. Download managers are interesting to consider since they will usually run in the background, either downloading (useful) or being idle (not useful). Games on the other hand are a source of great power consumption, since the utilisation of graphic cards raise the power consumption of the device. Like games, processes that are running for multimedia purposes are in need of more resources and lead to greater power consumption. Like the download category, the messaging category contains software that usually sits in the background either running a communication or being idle. Although analysing network power consumption is not important to the project, the network category provides software applications that can give interesting details. After browsers and messaging software, office and programming contain the most used software applications. They are usually used for work purposes. Since most of the participants are either studying or working in an IT environment, the category is named programming, although it contains some software used for other purposes (such as archaeology). Remote access software provides connectivity over networks to other places, devices or servers. These applications also denote that additional power is used over the network and by the devices that are being accessed. The system category contains services which are run in support to the applications opened by the user. It also includes installers, updaters and software which belong to the operating system, such as system management services. The virtual machine category is for processes which are related to virtual machines which are installed on the system, which leads to greater power consumption. Finally, the viruses category contains processes which are running threats to the system.

The analysis was conducted for each user in particular. The starting point was to sort out the data entries and to remove the entries which were considered unnecessary, such as system generated processes. The analysis targeted the four main values for each process entry: focused time, unfocused time, active time, inactive time. Using the periods of time registered in the power states database, the process entries were split accordingly into groups. Each group was then analysed based on the agreed assumptions. Changes were made to the values, more specifically were deemed appropriate the inactive value was changed to match the focused value. Also each user was given behaviour ratings which consisted of comparing the focused values of the applications accessed by the user to the total amount of time for which the trial was conducted (ratings are more appropriate here due to the possible errors in calculating exact percentages).

4.4 Data analysis

By analysing the data for each user in particular, user profiles have been created. Data has been collected on each user by analysing only the significant part of processes. This means that various processes have been removed from the list in order to make the data clearer. Such processes involve system processes and “garbage” processes (here garbage represents processes without important values for focused, unfocused, active and inactive). Each user profile will be presented along with interesting findings followed by the analysis of the user behaviour.

User 1 generally has many applications installed that start every time the system boots up. This means that power consumption is greater due to these applications. However, most of them are running in the user’s benefit. The email client and the download client are kept running in the background in order for the user to access them whenever necessary (he collected data shows that they are accessed). On the other hand there are many services for different software applications running in the background which are never accessed by the user (such as trays, update services, and other services kept running for quick access). Overall the user spends a significant time working on the machine, mostly using remote desktop connection software and network software. The data shows that the user has long periods of time for which he leaves the computer, sometimes to allow data transfers to happen (through the remote connections), but sometimes with no purpose, therefore consuming power with no use.

User 2 spends most of his time on the Internet. The data shows several times when the user performed window switching, particularly when using an Internet browser and writing tools. The user is also active in terms of multimedia and messaging. In terms of background processes, there are much less then for the previous user, which means smaller power consumption. Generally applications are only used when needed and there is a small amount of “garbage” processes.

User 3 spends most of his time on the Internet as well, but compared to user 2, the amount of inactive time is higher. Other activities include playing games and working on special applications designed for archaeology surveys. From the gathered data it is obvious that the user leaves applications running for long periods of time, in some situations leaving the computer without closing it. Since the applications in question do not point towards a specific purpose (mostly Internet browsing), the user has larger periods of inactivity and useless power consumption.

User 4 spends most time reading (probably doing research or studying). Windows switching is again very obvious between Internet browsers and document reading applications. There are no obvious background service processes that are running without use, and generally the computer usage looks tidy, meaning that the user only undertakes a small set of tasks and does not leave applications unattended.

As stated previously, the data gathered from user 5 will not be used in the analysis.

User 6 works most of the time on image processing. This was an interesting case, because during the image processing an application called reactivision is used. This application is a computer vision framework for tracking marking on physical objects. This means that during the task, the process running this application was focused, but because the user was interacting with the object the application was processing, the computer was not receiving any input from the user (the tracking application can only track mouse and keyboard input). Therefore the ratio between active and focused time for the user was much lower compared to others. After the data was analysed and changed, the ratio adjusted towards the normal parameters.

User 7 can be characterised as using the computer for leisure purposes. Most of the activity of the user consisted of browsing the Internet and accessing media on the computer. The user has one of the highest degrees of inactivity in the user trials (mostly due to the video watching). This is a good example of how the assumptions about accessing media are needed in order to get closer to a true ratio between active time and focused time.

User 8 can be characterised as being very untidy in terms of computer usage. At any given time there are a large number of applications opened, usually in many instances, such as multiple word documents (multiple web browser instances is common for all users). Overall it is the best example of unnecessary power consumption through direct user behaviour (user is knowingly idle without turning the computer off).

It is important to note that user 9 is the same person as user 8. The differences in the user trial are that the data gathered for user 8 is in a home environment on PC computer, while the data gathered for user 9 is in a work environment on a laptop. The behaviour of the user is still untidy, but the difference is that the inactive time is reduced drastically. Applications are still kept open in multiple instances, especially the word processing applications, which leads to unnecessary power consumption. However in a working environment it isn’t unusual for such behaviour. Similar to user 1, who also belonged to a working environment, user 9 constantly keeps communication applications (email client, messaging client) opened, which is where some of the inactivity is recorded by the tracking application.

User 10 provided data which was harder to analyse. Compared to the other users who had better opportunities for the assumptions to be applied, here the data was more ambiguous. The tasks undertaken were simple, without obvious window switching, so both the initial ratio and the final ratio were smaller than average. This meant that the user provided a high level of inactivity.

User 11 was similar to user 10, because the tasks undertaken were simple, mainly Internet browsing and document viewing (possibly studying purposes). Changes were made in regards to messaging and media viewing.

The data gathered from user 12 was the most interesting to observe and analyse due to its broad spectrum. From graphics editors to programming IDEs to remote connection software, the tasks undertaken by the user were very different. This made the user the most active (based on the unchanged data). By applying the assumptions the changed data showed that the user was still one of the most active. Because of the range of applications used, there were also various services required by these applications which were running in the background which lead to higher power consumption.

User 13 was the second user for which the tracking application had issues. Because the system was quite old, the tracking application crashed due to the previously mentioned error. Most of the user's activity regarded document editing and accessing multimedia content. It is interesting to note that the user had the best improvement in terms of ratio after applying the changes required by the assumptions towards multimedia accessing. This meant that the user was the most tidy in behaviour and the most active in terms of using applications. It was noticeable though that for some periods of time, the computer was left unattended, sometimes with applications running (noticed through the logon.scr process).

User 14 produced the less impressive results, because the main usage of the computer was Internet browsing. Compared to other users who focused on Internet browsing, this user did not do any other tasks. However the browsing was active which lead to a very high ratio (only on certain occasions was the browser left open unattended).

The last participant in the user trials, user 15 had similar results to user 14, but instead of focusing on Internet browsing, the focus was on gaming and multimedia content access. These two factors lead to a high activity ratio.

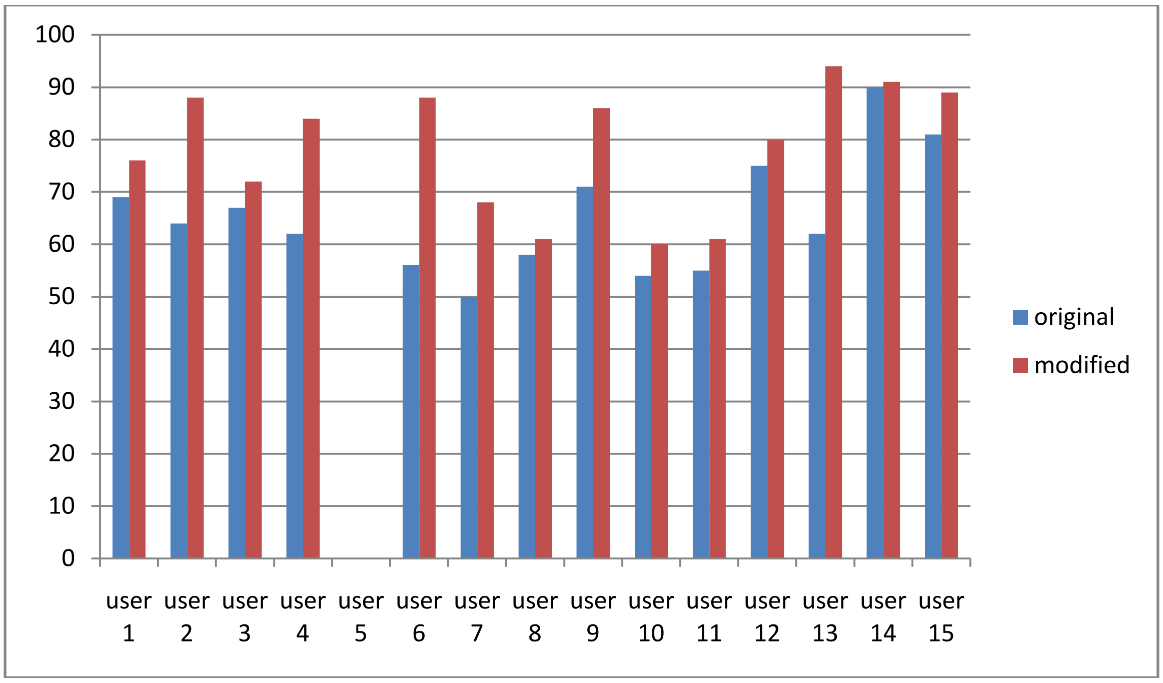


Table 4. Active time to focused time ratio

Table 4 represents the analysis made by applying the set of assumptions to the original data. It shows the ratio of active to focused time in terms of the original data gathered by the tracking application and the modified data obtained from the analysis. The most obvious improvement can be seen for user 6, which as stated previously used types of input different from mouse and keyboard which cannot be noticed by the tracking application. At the other end is the data for user 14, which predominantly involved Internet browsing. Due to the fact that the tracking application cannot access information about what the processes are doing, the inactivity of the user remained uncertain.

	background processes	idle time	power management	duration	sleep/hibernate	shut down
user 1	high	high	medium	18 days	9	3
user 2	low	low	good	14 days	19	15
user 3	low	high	bad	21 days	41	5
user 4	low	medium	good	12 days	17	6
user 5	n/a	n/a	n/a	1 day	n/a	n/a
user 6	medium	medium	good	18 days	6	34
user 7	low	high	good	14 days	8	24
user 8	medium	very high	bad	22 days	0	25
user 9	low	medium	medium	21 days	2	35
user 10	low	high	bad	15 days	21	3
user 11	medium	medium	medium	20 days	41	5
user 12	high	low	good	13 days	29	14
user 13	low	medium	good	18 days	0	40
user 14	low	low	good	16 days	22	6
user 15	medium	low	good	18 days	11	8

Table 5. Analysis of user behaviour in terms of power managing

Table 5 requires some additional explanations. The first column concerns background processes (the system processes are excluded). Some users have applications on their systems which start at system start-up and run in the background continuously. These applications can be plug-ins, add-ons, quick access system trays, updaters or messaging clients. Except the messaging clients which are directly accessed by the user, the others run usually without any purpose. By simply removing them from start-up, power consumption can be reduced (not drastically though). The column represents the amount of background processes representing such services that are running for each user.

The second column represents the amount of idle time calculated in terms of all the processes gathered from the user trials for each user. This was calculated by applying equations 2 and 3, where a task represented a group of processes for on power state entry (e.g. power on for the duration of 3 hours, all the focused processes in that hour were considered). As mentioned before, a rating classification was preferred to a percentage classification due to the large amount of information considered, as opposed to the smaller more focused amount of information considered for the active to focused ratio analysis.

The third column represents power management. This looks at the data in the States tables, which shows the durations for “power on” states and “power suspended” states. The “power off” states were easy to see as gaps between “power on” states. In combination with the last two columns which show the number of times “power off” and “power suspended” happened, the durations were analysed to get the results in the power management column. It is interesting to note that users 8 and 13 never had “power suspended” states. It is more interesting that they were the only two users who have PC computers instead of laptops. By comparison, user 13 has good power management, only using the computer when needed, and shutting it down when not, while user 8 has bad power management, developing a large amount of idle time and not shutting down the computer when not needed.

It is also interesting to note the users which prefer keeping their laptop in a sleep/hibernate mode, instead of shutting it down. Although at much lower levels, power is still consumed while the laptop is in sleep/hibernation mode. The mode is useful for short periods of inactivity, but for longer periods it causes unnecessary power consumption.

4.5 Usage patterns

In order to test power consumption, usage patterns need to be selected. The data analysis provided user profiles which can be used to select such usage patterns.

During the analysis, the highest uncertainty was when dealing with Internet browsers. Because the tracking application cannot track what the browsers are doing or how much power is consumed, this provides interesting usage patterns to be tested. The pattern will include common actions, such as downloading a file, watching a video online or surfing websites. Another test will be switching between an Internet browser and a document viewer to see if/how power consumption changes.

Another pattern derived from the analysis is accessing multimedia content on the computer, such as watching a video or listening to music. Here it will be interesting to see the differences between actively using the application and running it in the background.

Most of the participants in the trials have used one form of Internet messaging. Here it will be interesting to test the difference between communicating through classical input (mouse and keyboard) and communicating through a webcam.

A gaming pattern also needs to be tested. It is expected to take to power consumption to a high level due to usage of the video card and of a large amount of resources.

Some of the participants used remote connection clients. It is interesting to test this pattern in order to see how power consumption changes when the user interacts with another device through the remote connection.

Downloading or transferring files represent another interesting pattern. This can be achieved either by using a download client or by connecting a memory device to the computer and directly transferring the files.

4.6 Linking power consumption to process use

After the user trials were finished, the participants were asked to give some details that are needed for this part of the analysis.

	device	**brand**	**os**	**sleep or hibernate**	**monitor shutdown**
u1	laptop	Acer Travelmate 5744G	windows 7 ultimate	sleep	10 min
u2	laptop	Toshiba Satellite L550	windows 7 professional sp1	sleep	10 min
u3	laptop	Dell XPS L511	windows 7	hibernate	15 min
u4	laptop	HP Pavilion dv6	windows 7	sleep	n/a
u5	virtual machine	n/a	windows xp	n/a	n/a
u6	laptop	HP dv7-6150ev	windows 7 sp1	sleep	5 min
u7	laptop	HP Pavilion dm4	windows 7 home premium sp1	sleep	5 min
u8	pc	n/a	windows 7 home premium sp1	n/a	20 min
u9	laptop	HP Presario CQ57	windows 7 home premium sp1	hibernate	15 min
u10	laptop	Sony VCPEB11	windows 7 sp1	sleep	10 min
u11	laptop	Samsung P469	windows 7 professional sp1	sleep	5 min
u12	laptop	Dell Studio 1747	windows 7 sp1	sleep	10 min
u13	pc	n/a	windows xp professional sp3	n/a	20 min
u14	laptop	Lenovo b560	windows 7 home premium sp1	sleep	10 min
u15	laptop	Acer 5739G	windows 7 professional sp1	hibernate	5 min

Table 6. Systems used in user trials (with power options)

A PC desktop in general consumes more power than a laptop. It is interesting to note that both PC users, based on tables 5 and 6, do not have sleep or hibernate mode activated. Also they have the longest idle time for the monitor to go off. This shows that just by changing some settings on the operating system, power consumption can be reduced. However user satisfaction is also important, and perhaps the two users have specifically created these setups for their own satisfaction.

Energy Star is a US program that proposes energy efficient products and practices (Energy Star, 2012). Table 7 lists the specifications for all the laptops used during the user trials.

	Power in off	Power in sleep	Power in idle	Typical
Acer Aspire 5739	0.67 W	0.75 W	10.68 W	40 W
Acer Travelmate 5744	0.75 W	1.16 W	8.55 W	40 W
Dell Studio 1747	0.38 W	1.89 W	19.45 W	53 W
Dell XPS L511	0.35 W	1.29 W	7.54 W	53 W
HP dv7-6150ev	0.8 W	1.13 W	15.81 W	53 W
HP Pavilion dv6	0.64 W	0.78 W	9.04 W	40 W
HP Pavilion dm4	0.68 W	1.13 W	10.7 W	40 W
HP Presario CQ57	0.73 W	1.13 W	13.84 W	53 W
Lenovo b560	0.48 W	0.82 W	9.74 W	53 W
Toshiba Satellite L550	0.44 W	1.06 W	7.61 W	40 W
Sony VCPEB11	0.47 W	1.28 W	13.13 W	40 W
Samsung P469	0.64 W	1.22 W	16.15 W	53 W

Table 7. Registered power consumptions for participant laptops from Energy Star

Based on the Energy Star ratings, some of these laptops are category A (typical power consumption of 40 W), the others are category B (typical power consumption 53W). The Energy Star classification also contains category C, but none of the participants had a laptop from this category.

The laptop which will be used for the power consumption experiment based on the discussed patterns has the detailed specification in Table 8. The table contains the ratings taken from Energy Star as well as the ratings recorded by using the monitoring device. It is important to note that the Energy Star ratings are recorded by using a new laptop, which does not have any software installed except for the default software. The current recorded ratings are higher in value due to the age of the laptop and the settings.

Model	Toshiba Satellite T135	
OS	Windows 7 professional SP1	
Processor	Intel Centrino Duo T5200 @ 1.60GHz	
RAM	2.50GB	
	Energy Star ratings	**Watts Up Meter ratings**
Power in off	0.5 W	0.8 W
Power in sleep	0.66 W	1.5 W
Power in idle	7.91 W	19.5 W
Typical	40 W	48 W

Table 8. Laptop specifications

The device which was used to monitor power consumption is a “Watts Up? PRO”. The meter monitors power consumption and sends the data through a USB cable to the computer, where the data is displayed as a table and a graph (Watts Up, 2012).

Image 2. Test setup with Watts Up Meter and Toshiba Satellite T135

The first tested pattern was the Internet browser patter. Starting from idle (no processes opened by the user), the first step was to open a web browser (firefox.exe) and to watch a video. The graphs created by the meter are not very clear because the meter displays from time to time jumps in power consumption (usually when new visual objects appear on the laptop screen). However by analysing the tables produced, mean values for power consumption were produced. The next step was to access a news website (without any video, just text). That was followed testing window switching between the web browser and various document viewers. It is important to note that at any time, if the user scrolls quickly through a document, the power consumption jumps towards the maximum value. The last test was to download a file using the web browser. It produced surprising results, because it was expected to raise power consumption, however it didn’t.

	video	text	window switching	multiple documents	download
browser	27-30 W	23-25 W	35-40 W	35-40 W	27-30 W

Table 9. Internet browsing patterns (average values for 5 tests)

The next tested pattern was the multimedia patter. The first test was to play a music file using a media player. The song was 4 minutes long, so the test was to have the media player focused for minutes 1 and 3 and unfocused (minimised) for minutes 2 and 4. This time the graph clearly shows the difference between the two types of playing. When the media player was focused, the consumption averaged 25 W, when it was unfocused, it averaged 24 W. Of course when changing between focused and unfocused there are spikes in the power consumption.

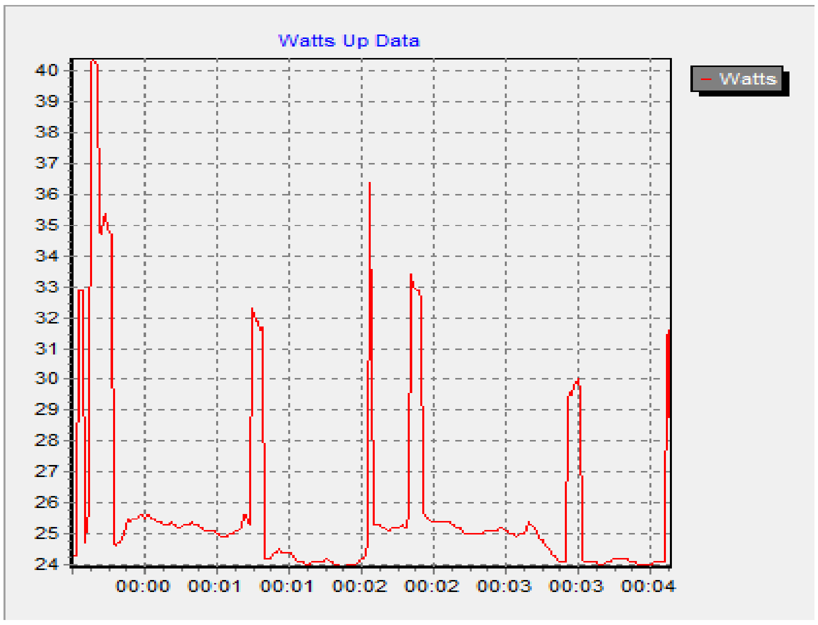


Image 3. Graph showing the difference between focused and unfocused music player

Testing the media player for playing a video was only done while focused (no point of having the video unfocused). Throughout the test, the consumption remained almost constant, ranging between 35 and 37 W.

The downloading pattern provided curious results. Similar to the Internet browser download test, where the power consumption was not as high as expected, this new test also proved to consume a small amount of power. The download speed was constantly around 300kB/s and the power consumption only went up to about 26 W. On average it stayed around 25 W. On the other hand the data transfer through USB from an external hard drive constantly consumed 38-39 W. This transfer was however done at speeds of 30MB/s.

Unfortunately, the gaming pattern could not be tested, because the video card installed on the laptop does not support new games or even some of the old games. However, when accessing a free gaming website with small flash games, the power consumption oscillated between 35W and 44W. It is therefore obvious that games will consume power close to the maximum limit, especially because of the graphics.

The remote connection client pattern was tested by connecting to the University Computer Science server (snowy). This meant remotely connecting to a Linux platform. The results showed that working on the remote connection leads to the same power consumption as working directly on the laptop. The consumption spiked when a web browser was accessed and when a video was played. Graphics provided to be the most important factor in terms of rising consumption. While not doing tasks which involved graphics, the consumption remained low, around 25 W.

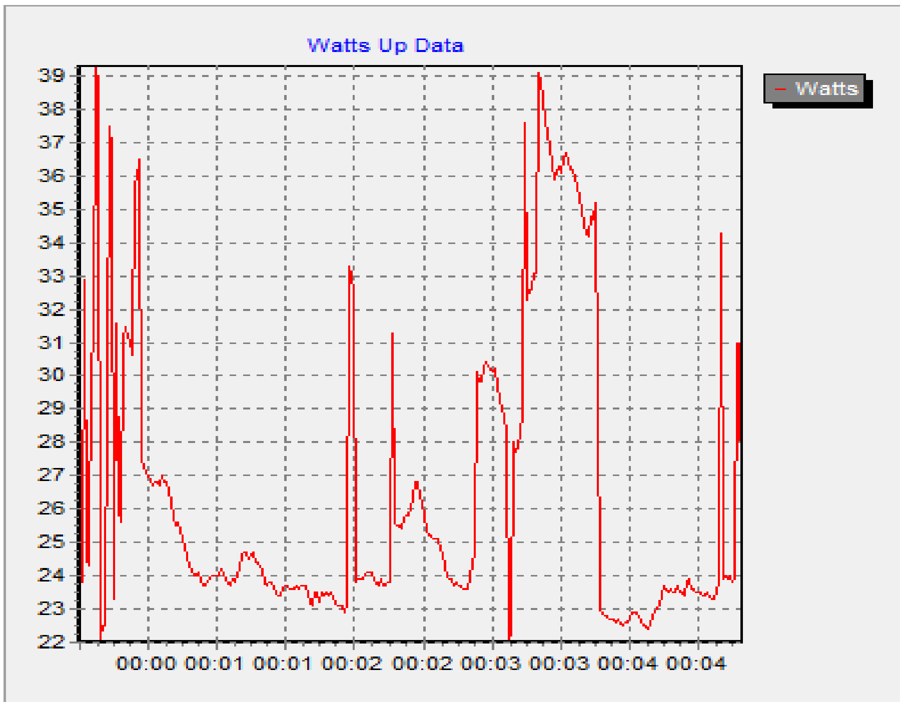


Image 4. Graph showing spikes in consumption when connecting through ssh to the remote computer, when accessing online video content, and finally when closing the connection

The messaging pattern test gave some important findings. It contained three stages; the first was text messaging, in which the power consumption levels remained normal. The second one was audio messaging (e.g. using headphones and a microphone). This part was expected to produce larger consumption, but it didn't. The last part was the most important, testing messaging through a webcam. This increased the consumption considerably.

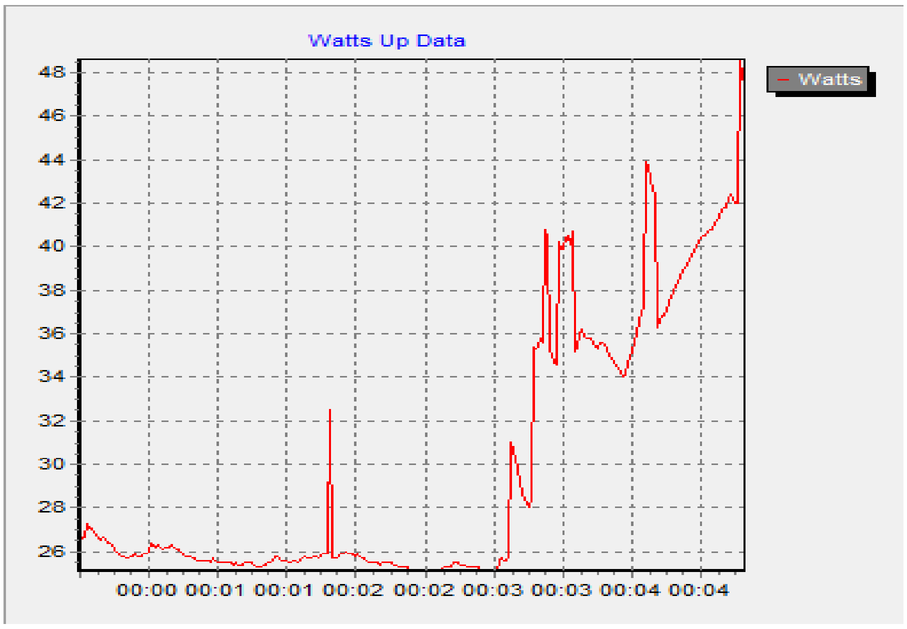


Image 5. Graph showing the first two stages of the test creating similar power consumptions, while the last stage increases consumption considerably due to the addition of the webcam

power in off	0.8 W	power with browser (download)	30 W
power in sleep	1.5 W	window switching	35 W
power in idle (monitor off)	14.5 W	power with music	25 W
power in idle	19.5 W	power with video	35 W
power in no input	22 W	power with download	25 W
power maximum	48 W	power with usb transfer	38 W
power with browser (text)	25 W	power with gaming	45 W
power with browser (video)	30 W	power with webcam	42 W

Table 10. Full set of ratings for the laptop recorded with Watts Up Pro

Since the laptops that were used in the trials have similar ratings given from Energy Star, it can be assumed that the ratings in Table 10 can be applied to them (provided that they are not brand new). The final classification that this analysis looks at is the list of patterns found in each of the user's gathered data. The power consumption values in Table 11 are given as averages (for example the download value is increased for higher speed downloads, while the messaging value is considered with a webcam, since most laptops have webcams).

patterns/users		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
browsers	~30 W	x	x	x	x		x	x	x	x	x	x	x	x	x	x
window switching	~35 W	x			x		x		x	x	x	x	x	x		
music	~25 W			x					x				x			
video	~35 W	x	x				x	x				x	x	x		x
download	~30 W	x	x					x						x		x
gaming	~45 W	x		x												x
messaging	~40 W	x	x	x	x		x	x	x	x	x	x	x	x	x	x
remote connect	~30 W	x	x				x	x			x	x	x			

Table 11. Users and patterns

Browsers and Internet messaging were used by all participants. In terms of averages, user 15 had the highest consumption average for the patterns, while user 8 had the lowest.

5 Conclusion

Power consumption depends on many factors, user behaviour being one of the most important and decisive. Through raising awareness, user behaviour can improve; therefore power consumption can be reduced. By simply changing settings on a computer or laptop, power consumption can be improved.

The uncertainty behind user behaviour affecting power consumption will be an important question for the decades to come. Simply by looking at the power consumption data gathered through the monitoring device, it is obvious that there are many factors to consider and that it is probably impossible to consider all of them.

5.1 Critical evaluation

The tracking application developed for this paper aimed to reduce uncertainty by looking at processes in terms of focused/unfocused and active/inactive. The gathered data helped with understanding user behaviour, but it proved to be quite difficult to analyse. Due to the fact that it was not possible to gather data about power consumption through the application the analysis proved to have many unknown variables. Gathering data that is 100% correct is not possible. Even though nowadays projects try to track the user's activity through sensors and other measures, the inability to predict a user's behaviour brings the need for assumptions and for the creation of user profiles.

The project tried to create these user profiles formed of patterns based on the data gathered by the tracking application, an application that can be improved to provide better results. The fact that the final deployment of the application missed an aspect of the design (not storing the list of processes as new for each session) proved to be a small setback in the data analysis. The results of the analysis were therefore more ambiguous than expected.

In terms of the power consumption measurement, the results are interesting and in some parts even unexpected. Testing patterns and ideas on the same computer does not give results for all the other computers, but it provides a benchmark for similar experimentation.

In the end, the project objectives were reached, even though in some places the results could have been much better. User behaviour is a vast area which needs to be researched more. A tracking application which monitors processes and user input cannot give full details about user behaviour, but it can give enough details to create general user profiles and patterns. Even though the number of participants to the user trials was not large, they proved to be different from each other through various actions and patterns.

By running experiments based on user patterns, the project achieved the aim to allocate power use to PC applications and to user behaviour. User behaviour will remain uncertain and power consumption will remain unpredictable, but the near future can only bring better understanding.

5.2 Future work

In terms of the tracking application there are many ideas and designs which can be applied. The first improvement can be the missing function to treat each “computer usage session” on its own. This would make the data analysis easier and will provide better results. “Garbage” data is something that the tracking applications needs to handle better, perhaps a filter can be used to sort processes into necessary and unnecessary. Other improvements can be made in terms of detecting activity, such as detecting not only user input, but also computer output. The idea here would be to detect audio or video output, which is however a very difficult implementation. The application could also track window titles by assigning them values so that they remain anonymous.

It is important to have the user trials “in-the-wild”, so that the data gathered is authentic. However at the end of the trials, a questionnaire can be used to solve some of the uncertainties. One idea would be to present the assumptions to the users and to ask which they agree on. Another idea would be to ask the users of what they think about their own behaviour when it comes to using a computer.

For the power consumption monitoring, more patterns can be easily identified, however due to the possibilities of the laptop used for the monitoring, they were not possible. Future additions to the patterns can be using virtual machines, accessing email locally on the machine, running applications that are resource intensive, such as games, image, audio and video editors.

6 Reference list

Beauvisage, T. (2009) ‘Computer usage in daily life’ In *Proceedings of the 27th international conference on Human factors in computing systems.* Held April 4-9 2009 in Boston, USA

Barata, G., Nicolau, H. and Goncalves, D. (2012) ‘AppInsight: What have I been doing’ In *Proceedings of the International Working Conference on Advanced Visual interfaces.* Held May 22-25 in Napoli, Italy

Brdiczka, O. (2010) ‘From documents to tasks: deriving user tasks from document usage patterns’ In *Proceedings of the 15th international conference on Intelligent user interfaces.* Held February 7-10 in Hong Kong, China

Tak, S. and Cockburn, A. (2009) ‘Window watcher: a visualization tool for understanding windowing activities’ In *Proceedings of the 21st Annual Conference of the Australian Computer-Human interaction Special interest Group.* Held November 23-27 in Melbourne, Asutralia

Kim, J.H., Gunn, D.V., Schuh, E.,Phillips, B., Pagulayan, R.J. and Wixon, D. (2008) ‘Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems’ In *Proceedings of the 26th annual SIGCHI conference on Human factors in computing systems.* Held April 5-10 2008 in Florence, Italy

Saito, R., Kuboyama, T., Yamakawa, Y. and Yasuda, H (2011) ‘Understanding user behaviour through summarization of window transition logs.’ In *Databases in Networked Information Systems 7th International Workshop.* Held December 12-14 2011 in Aizu-Wakamatsu, Japan

Shen, J., Li, L. and Dietterich, T.G.’Real-time detection of task switches of desktop users’ In *Proceedings of the International Joint Conference on Artificial intelligence.*

Tarzia, S.P., Dinda, P.A., Dick, R.P. and Memik, G. (2010) ‘Display power management policies in practice.’ In *Proceedings of the 7th international conference on Autonomic computing.* Held June 7-11 2010 in Reston, USA

Teehan, P., Kandlikar, M. and Dowlatabadi, H. (2010) ‘Estimating the changing environmental impacts of ICT-based tasks: A top-down approach.’ In *IEEE International Symposium on Sustainable Systems and Technology.* Held May 17-19 2010 in Arlington, USA

Da Costa, G. and Hlavacs, H. (2010) ‘Methodology of measurement for energy consumption of applications’ In *Proceedings of the 11th IEEE/ACM International conference on Grid Computing.* Held October 25-28 2010 in Brussels, Belgium

Smith, J.W., Khajeh-Hosseini, A., Ward, J.S. and Sommerville I. (2011) ‘CloudMonitor: Profiling applications and predicting power usage’ In *Proceedings of the IEEE 5th International Conference on Cloud Computing.* Held June 24-29 in Honolulu, USA

Kansal, A., Zhao, F., Liu, J., Kothari, N. and Bhattacharya, A.A. (2010) ‘Virtual machine power metering and provisioning.’ In *Proceedings of the 1st ACM symposium on Cloud Computing.* Held June 10-11 2010 in Indianapolis, USA

Bryman, A. and Bell, E. (2007) *Business research methods.* New York: Oxford University Press

W3Schools (2012) *OS Statistics* [online] available from <http://www.w3schools.com/browsers/browsers_os.asp>

MSDN (2012) *Walkthrough: Calling Windows APIs (Visual Basic)* [online] available from <http://msdn.microsoft.com/en-us/library/172wfck9.aspx>

SQLite (2012) *SQLite Home Page* [online] available from <http://www.sqlite.org/>

Energy Star (2012) *Energy Star* [online] available from <http://www.energystar.gov/>

Watts Up (2012) *Watts Up?* [online] available from <https://www.wattsupmeters.com/secure/index.php>

Appendix A: Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Data.SQLite;
using System.Data;
using Microsoft.Win32;

namespace ActiveWin
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            int init = 0;

            //create/connect to database
            SQLiteConnection sql_con1 = new SQLiteConnection("Data Source=" +
                System.Windows.Forms.Application.LocalUserAppDataPath + "\\data.db3");
            sql_con1.Open();
            SQLiteCommand sql_cmd1 = new SQLiteCommand(sql_con1);

            //create Settings table
            sql_cmd1.CommandText = "CREATE TABLE IF NOT EXISTS Settings (crt INTEGER, name VARCHAR(50), value
INTEGER," +
                "PRIMARY KEY (crt))";
            sql_cmd1.ExecuteNonQuery();

            SQLiteDataAdapter sqlda1 = new SQLiteDataAdapter("SELECT * FROM Settings", sql_con1);

            DataSet ds1 = new DataSet("xx");
            sqlda1.FillSchema(ds1, SchemaType.Source, "Settings");
            sqlda1.Fill(ds1, "Settings");

            //if empty, initialise table
            if (ds1.Tables[0].Rows.Count == 0)
            {
                sql_cmd1.CommandText = "INSERT INTO Settings (crt,name,value) VALUES (1,'running',0)";
                sql_cmd1.ExecuteNonQuery();
                sql_cmd1.CommandText = "INSERT INTO Settings (crt,name,value) VALUES (2,'initialized',0)";
                sql_cmd1.ExecuteNonQuery();
            }

            DataTable dt1 = ds1.Tables["Settings"];

            //check to see if application was left running, if yes start application automatically
            foreach (DataRow row in dt1.Rows)
            {
                if ((row["name"].ToString().CompareTo("running") == 0) && (Convert.ToInt32(row["value"]) == 1))
                {
                    init = 1;
                }

            }

            SQLiteDataAdapter sqlda2 = new SQLiteDataAdapter("SELECT * FROM Settings", sql_con1);

            DataSet ds2 = new DataSet("xxx");
            sqlda2.FillSchema(ds2, SchemaType.Source, "Settings");
            sqlda2.Fill(ds2, "Settings");

            DataTable dt2 = ds2.Tables["Settings"];

            using (SQLiteTransaction sqlt2 = sql_con1.BeginTransaction())
            {
                foreach (DataRow row in dt2.Rows)
                {
                    //if not initialised (first run of the app) add to registry key
                    if ((row["name"].ToString().CompareTo("initialized") == 0) && (Convert.ToInt32(row["value"]) ==
0))
                    {
                        RegistryKey rk =
Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true);

                        rk.SetValue("ActiveWin", Application.ExecutablePath.ToString());

                        row.BeginEdit();

                        row["value"] = 1;
                        row.EndEdit();
                    }
                }

                SQLiteCommandBuilder sqlcb2 = new SQLiteCommandBuilder(sqlda2);
                sqlda2.Update(ds2, "Settings");

                sqlt2.Commit();
            }

            //run application window
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new ActiveWin(init));
        }
    }
}

namespace ActiveWin
{
    partial class ActiveWin
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
                this.icon.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.run = new System.Windows.Forms.Button();
            this.stop = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // run
            //
            this.run.Location = new System.Drawing.Point(107, 88);
            this.run.Name = "run";
            this.run.Size = new System.Drawing.Size(75, 23);
            this.run.TabIndex = 0;
            this.run.Text = "run";
            this.run.UseVisualStyleBackColor = true;
            this.run.Click += new System.EventHandler(this.run_Click);
            //
            // stop
            //
            this.stop.Location = new System.Drawing.Point(107, 174);
            this.stop.Name = "stop";
            this.stop.Size = new System.Drawing.Size(75, 23);
            this.stop.TabIndex = 1;
            this.stop.Text = "stop";
            this.stop.UseVisualStyleBackColor = true;
            this.stop.Click += new System.EventHandler(this.stop_Click);
            this.stop.Enabled = false;
            //
            // ActiveWin
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(284, 262);
            this.Controls.Add(this.stop);
            this.Controls.Add(this.run);
            this.Name = "ActiveWin";
            this.Text = "Form1";
            this.ResumeLayout(false);
        }

        #endregion

        private System.Windows.Forms.Button run;
        private System.Windows.Forms.Button stop;
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SQLite;
using System.Runtime.InteropServices;
using System.Diagnostics;
using System.Threading;

namespace ActiveWin
{
    public partial class ActiveWin : Form
    {
        [DllImport("user32.dll")]
        static extern IntPtr GetForegroundWindow();

        [DllImport("user32.dll")]
        private static extern Int32 GetWindowThreadProcessId(IntPtr hWnd, out uint lpdwProcessId);

        [DllImport("user32.dll")]
        static extern bool GetLastInputInfo(ref LASTINPUTINFO plii);

        public struct LASTINPUTINFO
        {
            public uint cbSize;
            public uint dwTime;
        }

        Process currentProcess;

        string format = "dd-MM-yyyy HH:mm:ss";

        double elapsed = 0;

        string sactive = "";
        double active = 0;

        string sinactive = "";
        double inactive = 0;

        string sfocus = "";
        double focus = 0;

        string sunfocus = "";
        double unfocus = 0;

        int count = 0;

        private NotifyIcon icon = new NotifyIcon();

        private ContextMenu cm = new ContextMenu();

        int closeVar = 0;

        public ActiveWin(int init)
        {
            InitializeComponent();

            //initialise tray icon
            icon.Icon = new Icon(GetType(), "Icon2.ico");
            icon.Visible = true;

            //initialise tray icon menu
            cm.MenuItems.Add(0, new MenuItem("Show", new System.EventHandler(Show_click)));
            cm.MenuItems.Add(1, new MenuItem("Hide", new System.EventHandler(Hide_click)));
            cm.MenuItems.Add(2, new MenuItem("Exit application", new System.EventHandler(Exit_click)));

            icon.ContextMenu = cm;

            //if running at shutdown, restaart automatically
            if (init == 1)
            {
                stop.Enabled = true;
                run.Enabled = false;

                Hide();

                count = 1;

                Thread t1 = new Thread(runProgram);
                t1.IsBackground = true;
                t1.Start();
            }
        }

        //run button
        private void run_Click(object sender, EventArgs e)
        {
            this.stop.Enabled = true;
            this.run.Enabled = false;

            count = 1;

            Thread t = new Thread(runProgram);
            t.IsBackground = true;
            t.Start();
        }

        //stop button
        private void stop_Click(object sender, EventArgs e)
        {
            this.run.Enabled = true;
            this.stop.Enabled = false;

            this.pauseProgram();
        }

        //show menu
        protected void Show_click(Object sender, System.EventArgs e)
        {
            this.Show();
        }

        //hide menu
        protected void Hide_click(Object sender, System.EventArgs e)
        {
            this.Hide();
        }

        //exit application menu
        protected void Exit_click(Object sender, System.EventArgs e)
        {
            closeVar = 1;
            this.Close();
        }

        //minimise to tray if app window is closed
        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            base.OnFormClosing(e);

            if ((e.CloseReason == CloseReason.UserClosing) && (closeVar == 0))
            {
                e.Cancel = true;
                this.Hide();
            }
        }

        private void runProgram()
        {
            //connect to db
            SQLiteConnection sql_con = new SQLiteConnection("Data Source=" +
                System.Windows.Forms.Application.LocalUserAppDataPath + "\\data.db3");
            sql_con.Open();

            SQLiteCommand sql_cmd = new SQLiteCommand(sql_con);

            //initialise Processes table
            sql_cmd.CommandText = "CREATE TABLE IF NOT EXISTS Processes (crt INTEGER, id INTEGER, name VARCHAR(200),"
+
                "focus DOUBLE, unfocus DOUBLE, active DOUBLE, inactive DOUBLE, start VARCHAR(100), end VARCHAR(100),"
+
                "closed INTEGER, PRIMARY KEY (crt));";
            sql_cmd.ExecuteNonQuery();

            //initialise States table
            sql_cmd.CommandText = "CREATE TABLE IF NOT EXISTS States (crt INTEGER, name VARCHAR(50), start
VARCHAR(100)," +
                "end VARCHAR(100), closed INTEGER, PRIMARY KEY (crt));";
            sql_cmd.ExecuteNonQuery();

            sql_cmd.CommandText = "INSERT INTO States (crt,name,start,end,closed) VALUES
(NULL,'powerOn',?,'n/a',0);";

            SQLiteParameter para1 = sql_cmd.CreateParameter();
            sql_cmd.Parameters.Add(para1);

            DateTime now1 = DateTime.Now;

            para1.Value = now1.ToString(format);
            sql_cmd.ExecuteNonQuery();

            int init = 0;

            DateTime last = DateTime.Now;

            //check that app was not stoped by user
            while (count == 1)
            {
                Stopwatch sw = new Stopwatch();
                sw.Start();

                //get list of processes
                Process[] a = Process.GetProcesses();

                //get focused process
                currentProcess = GetActiveProcess();

                DateTime current = DateTime.Now;

                SQLiteDataAdapter sqlda4 = new SQLiteDataAdapter("SELECT * FROM States", sql_con);

                DataSet ds4 = new DataSet("xxxx");
                sqlda4.FillSchema(ds4, SchemaType.Source, "States");
                sqlda4.Fill(ds4, "States");

                DataTable dt4 = ds4.Tables["States"];

                using (SQLiteTransaction sqlt4 = sql_con.BeginTransaction())
                {
                    //change last entry in States table if uninitialised
                    if (init == 0)
                    {
                        init++;

                        foreach (DataRow row in dt4.Rows)
                        {
                            if (((Convert.ToInt32(row["closed"])) == 0) &&
(now1.ToString(format).CompareTo(row["start"]) != 0))
                            {
                                DateTime l = GetLastSystemShutdown();

                                row.BeginEdit();

                                row["end"] = l.ToString(format);

                                row["closed"] = 1;

                                row.EndEdit();
                            }
                        }
                    }
                    else
                    {
                        TimeSpan diff = current - last;

                        //check to see if sleep/hibernate mode was on
                        if (diff.TotalSeconds > 60)
                        {
                            foreach (DataRow row in dt4.Rows)
                            {
                                if ((Convert.ToInt32(row["closed"])) == 0)
                                {
                                    row.BeginEdit();

                                    row["end"] = last.ToString(format);

                                    row["closed"] = 1;

                                    row.EndEdit();
                                    sql_cmd.CommandText = "INSERT INTO States (crt,name,start,end,closed) " +
                                        "VALUES (NULL,'powerSuspended',?,?,1);";

                                    SQLiteParameter para2 = sql_cmd.CreateParameter();

                                    sql_cmd.Parameters.Add(para2);

                                    para1.Value = last.ToString(format);
                                    para2.Value = current.ToString(format);

                                    sql_cmd.ExecuteNonQuery();

                                    sql_cmd.CommandText = "INSERT INTO States (crt,name,start,end,closed) " +
                                        "VALUES (NULL,'powerOn',?,'n/a',0);";

                                    para1.Value = current.ToString(format);

                                    sql_cmd.ExecuteNonQuery();
                                }
                            }
                        }
                    }

                    SQLiteCommandBuilder sqlcb4 = new SQLiteCommandBuilder(sqlda4);
                    sqlda4.Update(ds4, "States");

                    sqlt4.Commit();
                }

                SQLiteDataAdapter sqlda2 = new SQLiteDataAdapter("SELECT * FROM Settings", sql_con);

                DataSet ds2 = new DataSet("x");
                sqlda2.FillSchema(ds2, SchemaType.Source, "Settings");
                sqlda2.Fill(ds2, "Settings");

                DataTable dt2 = ds2.Tables["Settings"];

                //change running setting to true
                using (SQLiteTransaction sqlt2 = sql_con.BeginTransaction())
                {
                    foreach (DataRow row in dt2.Rows)
                    {
                        if ((row["name"].ToString().CompareTo("running") == 0) && (Convert.ToInt32(row["value"]) ==
0))
                        {
                            row.BeginEdit();

                            row["value"] = 1;

                            row.EndEdit();
                        }
                    }

                    SQLiteCommandBuilder sqlcb2 = new SQLiteCommandBuilder(sqlda2);
                    sqlda2.Update(ds2, "Settings");

                    sqlt2.Commit();
                }

                SQLiteDataAdapter sqlda = new SQLiteDataAdapter("SELECT * FROM Processes", sql_con);

                DataSet ds = new DataSet("xx");
                sqlda.FillSchema(ds, SchemaType.Source, "Processes");
                sqlda.Fill(ds, "Processes");

                DataTable dt = ds.Tables["Processes"];

                using (SQLiteTransaction sqlt = sql_con.BeginTransaction())
                {
                    foreach (Process p in a)
                    {
                        foreach (DataRow row in dt.Rows)
                        {
                            //check database for valid process entry
                            if ((p.Id == Convert.ToInt32(row["id"])) &&
(p.ProcessName.CompareTo(row["name"].ToString()) == 0) &&
                                (Convert.ToInt32(row["closed"]) == 0))
                            {
                                row.BeginEdit();

                                //check if focused process
                                if ((p.Id == currentProcess.Id) &&
(p.ProcessName.CompareTo(currentProcess.ProcessName) == 0))
                                {
                                    //check if user is idle
                                    int idle = GetLastInputTime();

                                    sfocus = row["focus"].ToString();
                                    focus = Convert.ToDouble(sfocus);
                                    focus += elapsed;
                                    sfocus = focus.ToString();
                                    row["focus"] = sfocus;

                                    if (idle > 60)
                                    {
                                        sinactive = row["inactive"].ToString();
                                        inactive = Convert.ToDouble(sinactive);
                                        inactive += elapsed;
                                        sinactive = inactive.ToString();
                                        row["inactive"] = sinactive;
                                    }
                                    else
                                    {
                                        sactive = row["active"].ToString();
                                        active = Convert.ToDouble(sactive);
                                        active += elapsed;
                                        sactive = active.ToString();
                                        row["active"] = sactive;
                                    }
                                }
                                //update unfocused processes
                                else
                                {
                                    sunfocus = row["unfocus"].ToString();
                                    sinactive = row["inactive"].ToString();
                                    unfocus = Convert.ToDouble(sunfocus);
                                    inactive = Convert.ToDouble(sinactive);
                                    unfocus += elapsed;
                                    inactive += elapsed;
                                    sunfocus = unfocus.ToString();
                                    sinactive = inactive.ToString();
                                    row["unfocus"] = sunfocus;
                                    row["inactive"] = sinactive;
                                }

                                row["closed"] = 2;

                                DateTime time = DateTime.Now;

                                row["end"] = time.ToString(format);

                                row.EndEdit();

                                goto exitLoop;
                            }
                        }

                        //add new processes to database
                        using (SQLiteCommand sqlc = sql_con.CreateCommand())
                        {
                            sqlc.CommandText = "INSERT INTO Processes
(crt,id,name,focus,unfocus,active,inactive,start,end,closed)" +
                            " VALUES (NULL,?,?,0,0,0,0,?,'n/a',0)";
                            SQLiteParameter param1 = sqlc.CreateParameter();
                            SQLiteParameter param2 = sqlc.CreateParameter();
                            SQLiteParameter param3 = sqlc.CreateParameter();
                            sqlc.Parameters.Add(param1);
                            sqlc.Parameters.Add(param2);
                            sqlc.Parameters.Add(param3);

                            DateTime now = DateTime.Now;

                            param1.Value = p.Id;
                            param2.Value = p.ProcessName;
                            param3.Value = now.ToString(format);

                            sqlc.ExecuteNonQuery();
                        }

                    exitLoop:
                        continue;
                    }

                    //update closed values
                    foreach (DataRow row in dt.Rows)
                    {
                        if (Convert.ToInt32(row["closed"]) == 2)
                        {
                            row.BeginEdit();
                            row["closed"] = 0;

                            row.EndEdit();
                        }
                        else if (Convert.ToInt32(row["closed"]) == 0)
                        {
                            row.BeginEdit();

                            row["closed"] = 1;

                            row.EndEdit();
                        }
                    }

                    SQLiteCommandBuilder sqlcb = new SQLiteCommandBuilder(sqlda);
                    sqlda.Update(ds, "Processes");

                    sqlt.Commit();
                }

                last = DateTime.Now;

                System.Threading.Thread.Sleep(2000);

                sw.Stop();

                elapsed = sw.Elapsed.TotalSeconds;
            }
        }

        private void pauseProgram()
        {
            //interrupts the runProgram() loop
            count = 0;

            SQLiteConnection sql_con = new SQLiteConnection("Data Source=" +
                System.Windows.Forms.Application.LocalUserAppDataPath + "\\data.db3");
            sql_con.Open();

            SQLiteDataAdapter sqlda1 = new SQLiteDataAdapter("SELECT * FROM States", sql_con);

            DataSet ds1 = new DataSet("x");
            sqlda1.FillSchema(ds1, SchemaType.Source, "States");
            sqlda1.Fill(ds1, "States");

            DataTable dt1 = ds1.Tables["States"];

            using (SQLiteTransaction sqlt1 = sql_con.BeginTransaction())
            {
                foreach (DataRow row in dt1.Rows)
                {
                    if (Convert.ToInt32(row["closed"]) == 0)
                    {
                        row.BeginEdit();

                        row["closed"] = 1;

                        DateTime now = DateTime.Now;

                        row["end"] = now.ToString(format);

                        row.EndEdit();
                    }
                }

                SQLiteCommandBuilder sqlcb1 = new SQLiteCommandBuilder(sqlda1);
                sqlda1.Update(ds1, "States");

                sqlt1.Commit();
            }

            SQLiteDataAdapter sqlda2 = new SQLiteDataAdapter("SELECT * FROM Settings", sql_con);

            DataSet ds2 = new DataSet("xx");
            sqlda2.FillSchema(ds2, SchemaType.Source, "Settings");
            sqlda2.Fill(ds2, "Settings");

            DataTable dt2 = ds2.Tables["Settings"];

            //set running to false
            using (SQLiteTransaction sqlt2 = sql_con.BeginTransaction())
            {
                foreach (DataRow row in dt2.Rows)
                {
                    if ((row["name"].ToString().CompareTo("running") == 0) && (Convert.ToInt32(row["value"]) == 1))
```

```
                    {
                        row.BeginEdit();

                        row["value"] = 0;

                        row.EndEdit();
                    }
                }

                SQLiteCommandBuilder sqlcb2 = new SQLiteCommandBuilder(sqlda2);
                sqlda2.Update(ds2, "Settings");

                sqlt2.Commit();
            }
        }

        private static Process GetActiveProcess()
        {
            IntPtr handle = GetForegroundWindow();

            return handle != null ? GetProcessByHandle(handle) : null;
        }

        private static Process GetProcessByHandle(IntPtr handle)
        {
            try
            {
                uint processID;
                GetWindowThreadProcessId(handle, out processID);
                return Process.GetProcessById((int)processID);
            }
            catch
            {
                return null;
            }
        }

        static int GetLastInputTime()
        {
            uint idleTime = 0;

            LASTINPUTINFO lii = new LASTINPUTINFO();
            lii.cbSize = (uint)Marshal.SizeOf(lii);
            lii.dwTime = 0;

            uint envTicks = (uint)Environment.TickCount;

            if (GetLastInputInfo(ref lii))
            {
                uint lit = lii.dwTime;
                idleTime = envTicks - lit;
            }

            return ((idleTime > 0) ? (int)(idleTime / 1000) : 0);
        }

        public static DateTime GetLastSystemShutdown()
        {
            string sKey = @"System\CurrentControlSet\Control\Windows";
            Microsoft.Win32.RegistryKey key = Microsoft.Win32.Registry.LocalMachine.OpenSubKey(sKey);

            string sValueName = "ShutdownTime";
            byte[] val = (byte[])key.GetValue(sValueName);
            long valueAsLong = BitConverter.ToInt64(val, 0);
            return DateTime.FromFileTime(valueAsLong);
        }
    }
}
```

Appendix B: participation sheet

This is the read me file for the ActiveWin application.

IMPORTANT: Your participation in these user trials will be kept anonymous. All data gathered from you will be kept anonymous and will not be used in other projects or by other individuals. Your data privacy will not be affected by these user trials in any way.

The role of the application is to run in the background and gather some information about the computer.

After the application is installed, it needs to be started.

- Open the .exe file and you will see two buttons (Run and Pause).
- Click the run button.
- After that you can safely close the application window (by pressing the x button). This will not completely close the application, but it will leave it to run in the background (which is the whole idea).
- Notice on the bottom right of your taskbar there will be an orange icon which shows that the application is active.
- By right-clicking the icon, it will give the possibility to show/hide the application windows.
- IMPORTANT: By clicking "Exit application" on the icon, it will completely close the application. Do not exit the application unless told or unless the application is causing problems (which it shouldn't).
- The application does not consume resources and should run smoothly.
- Do not worry about closing the computer, as the application is built to close itself at shutdown and to restart itself when the computer starts again.
- IMPORTANT: If the application crashes, it will not cause problems to your operating system, it will only give an error message. Please contact me if so, and please start the application again, the data that it gathers will not get corrupted if the application crashes.
- After a few weeks I will ask you to send me a file called data.db3 (the file which contains all the gathered data). After you have sent me the file, you can safely exit the application completely and uninstall it from the add/remove programs menu.

Thank you for your cooperation, Andrei Mocanu

Appendix C: finalisation sheet

What to do:

- open the activewin application window (from the orange icon in your taskbar) by selecting show, click on stop, then on the orange icon click on exit application (this will completely stop the application)
- in windows explorer (any folder) go to organize/tools, then folder options, then view, then in the list click on show hidden files and folders
- for windows vista or 7, go to C:/Users/USERNAME/AppData/Local/Activewin/Activewin/1.0.0, there you will find a file named data.db3
- for windows XP, it is a similar path (you might need to browse through some more folders)
- send me the file named data.db3
- you can then uninstall the application from the add/remove programs menu
- in order to remove the application from the registry, go to Start, type regedit, and go to HKEY_CURRENT_USER, Software, Microsoft, Windows, CurrentVersion, Run; there will be an entry called ActiveWin, delete it

Additional things I would like to know about your computer (useful for my thesis):

- is it a laptop or a PC
- if it is a laptop, what laptop is it (e.g. Toshiba Satellite A135, Acer Aspire 5050, etc.)
- what Windows do you have installed (XP, Vista, 7 or 8)
- what service pack you have installed (if you know, if not it's ok)
- when you leave your computer, does it go to sleep or hibernate?
- after how much time does your display/monitor turn off?

If you have any questions or are unsure how to do any of the tasks, please contact me.