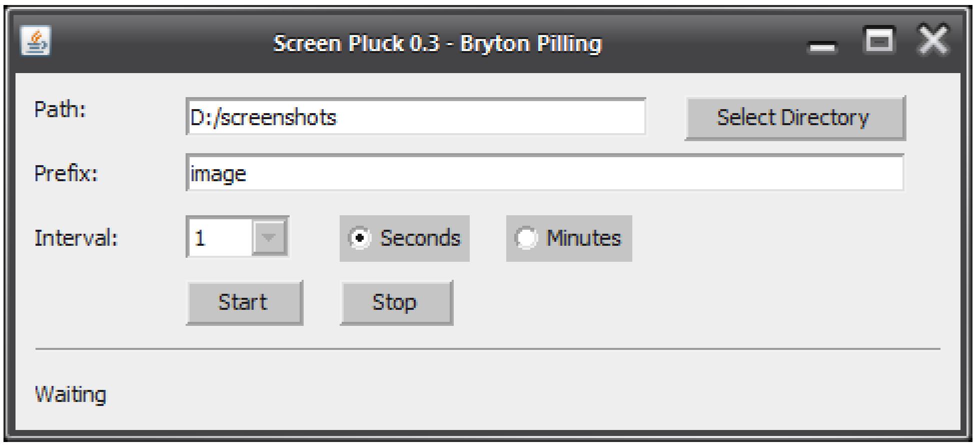
Screen Pluck 0.3 – Test analysis

This is a test analysis written by Rikard Edgren, version 0.4.
It is done for educational purposes, the idea is that it should provide an example of how to think in order to get the basis for testing a product well.

It is a very small application, that's why it is possible to go into detail also with the documentation. The thought processes are roughly the same as for more complex software, but also totally different.

Introduction

Screen Pluck is an application that takes screen captures at time intervals specified by user.


Screen Pluck 0.3 - Bryton Pilling

Path: D:/screenshots

Prefix: image

Interval: 1 (● Seconds, ○ Minutes)

Waiting

The user specifies file names and folder location, the software will add an increasing number.

Strategies for test ideas

I start with using the software benevolently, trying to understand the purpose, and see its capabilities.
I will get a lot of ideas about things to test, and I prefer to write them down in one-liner format.

To be sure I test the software well I make a model, because it helps me think about each ”object” and ”action”.

Next I will have a look at "Software Quality Characteristics" to see I didn't miss something important.

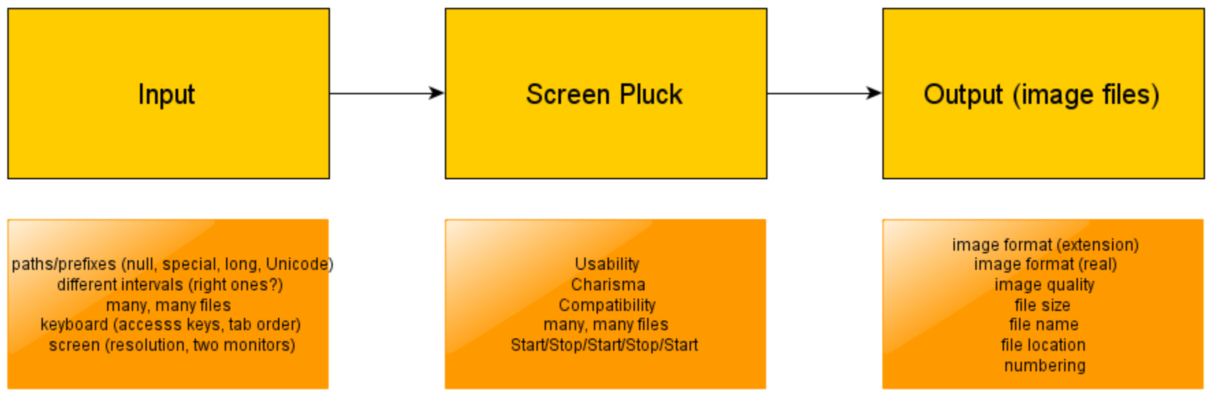
I do an SFDIPOT-analysis, where I can omit parts I already have considered (I usually start with SFDIPOT, but this time I did it later. You use what you want, in your preferred order, the important thing is that the result is an identification of things important to test, plus a good understanding.)

I conclude by considering risks that can be important.

As a complement I had a brief look at the source code.

A model

A simple model where I look at each "object" and their interactions.



For each thing in a model, you can ask yourself *"What if this one doesn't exist?"*
In this case it gave test ideas: run Screen Pluck without monitor, shut down Screen Pluck when running, Save to non-existing location.

Quality Characteristics

I look at the generic quality model http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf, pick the relevant ones, and try to make them as specific as possible for Screen Pluck. This is more difficult than it seems, and the core ingredients are an understanding of the software, plus experience from other software.

These quality characteristics seem most important for Screen Pluck:

Capability. *Test that the functions perform what is promised*

- *Completeness:* available time intervals should be the "right" ones.
- *Accuracy:* screen pluck each 10 seconds should create 100 images after 1000 seconds.
- *Interoperability:* could there be problems with "path vs. prefix" or "numbers vs. second/minute"?
- *Concurrency:* what happens if four Screen Plucks are run at the same time?
- *Data agnosticism:* test valid and invalid values for path and prefix

Reliability. *Can you trust the product in many and difficult situations?*

- *Stability:* run Screen Pluck over the weekend
- *Robustness:* test error handling for (very) invalid folders and file names
- *Stress handling:* test what happens if disk is full; test with very little available memory or CPU

Usability. *Heuristic evaluation of usability sub-categories.*

- *Intuitive:* is it easy to understand how to get started?
- *Learnability:* appropriately small program that is easy to learn and remember?
- *Operability:* correct tab order and access keys?
- *Interactivity:* is it easy to see if product is running or paused?
- *Control:* does the user feel in charge?
- *Clarity:* is terminology easy to understand for "everyone"?
- *Error handling:* look at error message for invalid entries.
- *Tailorability:* will Screen Pluck persist path and interval when restarted?
- *Accessibility:* can Screen Pluck be used by blind? (e.g. Microsoft Narrator)
- *Documentation:* can I get relevant Help about Screen Pluck?

Charisma. *Free software need charisma to reach an audience*

- *Professionalism:* is GUI trustworthy, is it appropriately good looking?
- *Directness:* let ten persons look at Screen Pluck and say how they like it

Security. *Can be ignored for Screen Pluck?*

Performance. *Is Screen Pluck fast enough?*
- *Capacity:* use an incredibly large monitor, and take one screen capture per second
- *Resource usage:* is reasonable amounts of memory and processor used?
- *Response time:* is perceived performance good?
- *Endurance:* can it run for weeks?

IT-bility. *Screen Pluck doesn't have an installer, but should be easy to handle*
- *System requirements:* test on computers without Java, or with too old Java version.

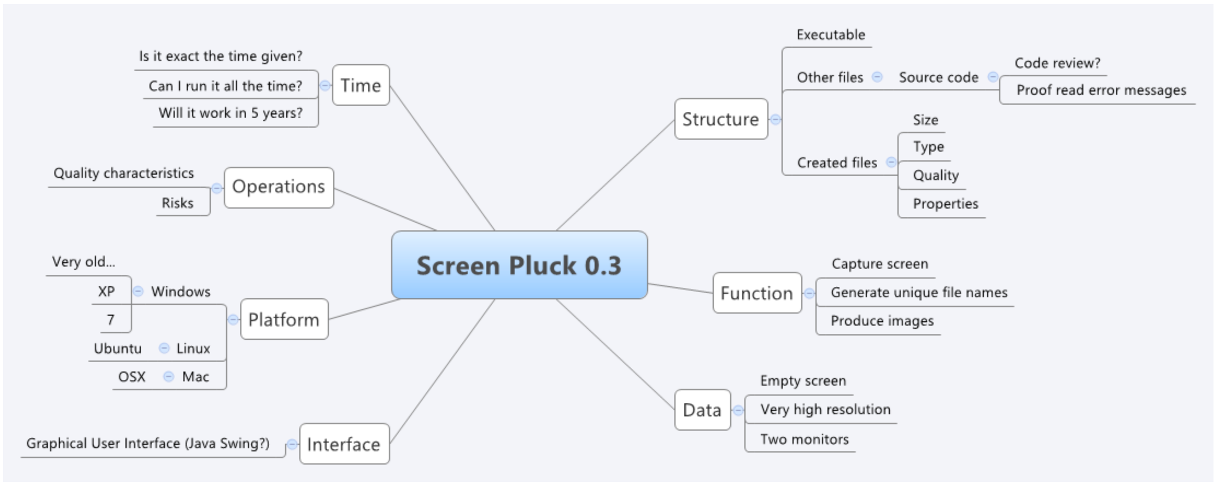
Compatibility. *How well does the product interact with software and environments?*
- *Hardware Compatibility*: find out how graphics card might matter.
- *Operating System Compatibility*: test on at least Windows, Linux and Mac (latest, and very old version)
- *Application Compatibility*: test with different kinds of screen "painting", e.g. full-screen games.
- *Configuration Compatibility*: test with Large DPI on Windows, and other special settings.

Supportability. *Can customers' usage and problems be supported?*
- *Identifiers:* does image properties state they were created by Screen Pluck?
- *Diagnostics:* is it possible to get details from a customer situation?
- *Troubleshootable:* is it easy to pinpoint an error (e.g. log file) and help?
- *Debugging:* can you observe internal states and variables if necessary?
- *Versatility:* is it possible to use Screen Pluck in other ways than intended?

Testability. *Will be easier to evaluate after testing has started.*
- *Traceability:* what should a log file contain?
- *Controllability:* if free-form intervals were available (10 captures per second), it would be easier to evaluate performance.

SFDIPOT model

A lot has already been covered, but there were some new ideas when thinking through Structure, Function, Data, Platform, Interfaces, Operations, Time:



What about computer hibernation mode?

Risks

I start by reading public information about the product at
http://sourceforge.net/projects/screenpluck/

> Description:
> A simple screenshot utility that takes screenshots at user selected intervals and saves them as png files to a directory selected by the user. Written in Java using the Netbeans IDE. Works on any modern operating system.
>
> Features:
> * Small and Light Weight

* Works on any operating System
* Takes Screenshots at timed intervals
* Images created are in .png format

I investigate keywords which gives me a few risks I think a product owner would agree with.
It is my understanding of this and other software that help me do this (an SFDIPOT model can also help.)

simple - risk that software isn't easy to use
screenshots - risk that complex "screens" aren't supported
user intervals - risk that frequency can't be adjusted as user wish
png - risk that images aren't saved in best way (quality, size, properties)
NetBeans – low risk that this component has serious problems in itself
any OS - risk that Screen Pluck won't function on some operating systems
lightweight - risk that installation and running isn't as smooth as one believes

At project page there are no reviews, but three "thumbs up".
There is no help or other product information accompanying the software (this could be seen as a problem in itself.)
There is a blog where the developer informs a little about new versions (0.4 is the latest, adding an "Overwrite" check box.)

I google "Screen Pluck" and mostly find it on download sites
I find other blogs from developer where he explains that the software was written when he wanted to document working with a building in Second Life.

Lateral thinking helped me identify this risk:
Risk that developer don't have the rights to distribute the code (if it is borrowed from other places.)

Test methods for important risks

Next step is to choose which risks to test, and how.
One can start with some kind of prioritization, and think about ways to test.

Prio	Risk	Test method
1	Screen capture doesn't match reality	Test basic functionality on may platform, with double monitors, and very high resolution.
2	Software is not easy to use for all	Do a heuristic evaluation of the identified usability characteristics. Start a reference group and interview them.
3	Not compatible on all platforms supporting Java, version 6	Test supported platforms (executed at the same time as risk 1)
4	Charisma	Let the reference group tell what they think after 5 minutes (no instructions)
5	Software is not stable and robust	Perform stress testing over a weekend.
6	General Public License violations	Read the code to notice interesting things. Run software that checks if code is ”borrowed”.
7	Other risks	Do exploratory risk-based testing on things not covered, and from other inspiration (competitors, SFDIPOT model, etc.)

The last one is very open, which also enables adding new ideas we come up with as we test.

Test ideas

Not everything can be tested, so the judgment about what to do is difficult and important.

These test ideas seem most important, and should be included, whatever method you have been using. In some kind of priority order:

1. Test that the created images reflects what is shown on screen
2. Test that vaild paths and prefix can be used
3. Test on supported operating systems
4. Test that interval time is at least reasonably correct
5. Evaluate usability
6. Investigate if image size and quality is appropriate.
7. Run Screen Pluck for a long time to evaluate reliability, endurance and resource usage.
8. Investigate if software has any charisma
9. Test error handling for various incorrect input.

All the other test ideas in this document can be treated as “run if they are fast and we have nothing better to do.”