Logan Montgomery

CSC362

Prof. McCartin-Lim

4/1/2021

The Unix Philosophy Reflection

Systems that utilize the 17 principles of design:

1. Rule of Modularity: Write simple parts connected by clean interfaces.

Mario, Design of Mario is an ‘object’ based system such that objects displayed on the screen have independent functions. This is an advantage because each object can be debugged individually without effect other objects.

1. Rule of Clarity: Clarity is better than cleverness.

Python enforces this well because much of the code looks very sudo code like it is typically clear what is happening.

1. Rule of Composition: Design programs to be connected to other programs.

ML forces this rule its design. Each function in ML takes only one input and gives one output. Function composition is hoe ML gets its power. This is nice because most functions tend to be similar and can be composed inside most other functions.

1. Rule of Separation: Separate policy from mechanism; separate interfaces from engines.

Any OS that uses a hardware extraction layer. The hardware and software both talk to this intermediate layer to make the correct connection between the software and hardware. This allows multiple different software to run on the same hardware.

1. Rule of Simplicity: Design for simplicity; add complexity only where you must.

Most electrical systems in homes strive to have simple designs. This makes it less likely to cause issues such as fires and makes a safer environment when the overall system is simple

1. Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.

This is similar to simple is better, another system that uses this a theatrical production, not a Hollywood movie, but a Broadway show. They tend to look big and dramatic but behind the curtain its mostly simple sytems that build up the entire show. This keeps the cost of production down.

1. Rule of Transparency: Design for visibility to make inspection and debugging easier.

This rule is seen in most vehicles. They are discoverable in that we can determine what all the switches and pedals do, and the dashboard allows for driver to see internal status such as temperature, rpm, fuel level, etc. These are important feedback for the drivers safety and convivence.

1. Rule of Robustness: Robustness is the child of transparency and simplicity.

Chat bots are robust under Unix’s definition. That is, most chat bots will operate with obscured and unexpected inputs. Well, maybe not csc 111 chat bots, but certainly the online chat bots abuse the principle of being robust to fake being human.

1. Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.
2. Rule of Least Surprise: In interface design, always do the least surprising thing.
3. Rule of Silence: When a program has nothing surprising to say, it should say nothing.
4. Rule of Repair: When you must fail, fail noisily and as soon as possible.
5. Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.
6. Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
7. Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
8. Rule of Diversity: Distrust all claims for “one true way”.
9. Rule of Extensibility: Design for the future, because it will be here sooner than you think.

Systems that don’t utilize the 17 principles of design:

1. Rule of Modularity: Write simple parts connected by clean interfaces.

WRK, Was not simple parts at all. WRK would be better because when bugs arise they could be fixed.

1. Rule of Clarity: Clarity is better than cleverness.

Java does not do this. Most java code is confusing. It might be clever to force all programs to be object oriented but it’s not always clear what’s happing. If java was clearer, it would be easier to write and maintain.

1. Rule of Composition: Design programs to be connected to other programs.

WRK does not do this. Many functions in WRK output very different data types. This makes it difficult to utilize functions within other parts of the code.

1. Rule of Separation: Separate policy from mechanism; separate interfaces from engines.

A system that does not do this is

1. Rule of Simplicity: Design for simplicity; add complexity only where you must.

IBM did not have this in mind when they paid their programmers by lines of code. This will inherently breed bad and complex code often with redundant lines. This makes code much longer and having a simple philosophy could improve the code greatly.

1. Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.

Like I mentioned up top, Hollywood productions are notorious for being over the top, expensive and complex. Typically, the most complicated way to do something is very expensive so we see the budgets for some modern films into multiple millions of dollars.

1. Rule of Transparency: Design for visibility to make inspection and debugging easier.

This may be a really dumb example but it really applicable. Doors. Doors are notoriously non transparent. Bad door designs are everywhere, we’ve all tried to pull a door that needed pushed or the other way around. The are hard to look at and determine whoch way to operate unless they are transparent which man aren’t. Look up Norman doors if you don’t believe this is a huge issue.

1. Rule of Robustness: Robustness is the child of transparency and simplicity.

Unix says robust is when the system operates under un expected conditions. Under this definition, many industrial machines are not robust. Many machines have an over abundance of safety switches which will simply terminate any process if there is some unexpected condition, such as someone’s hand in the way. For this, these systems might actually be worse if Unix considered them “robust”

1. Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.
2. Rule of Least Surprise: In interface design, always do the least surprising thing.
3. Rule of Silence: When a program has nothing surprising to say, it should say nothing.
4. Rule of Repair: When you must fail, fail noisily and as soon as possible.
5. Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.
6. Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
7. Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
8. Rule of Diversity: Distrust all claims for “one true way”.
9. Rule of Extensibility: Design for the future, because it will be here sooner than you think.