Subject: Fwd: writing good software is like poetry... or a good song
From: Joseph Picone <picone@temple.edu>
Date: 12/18/22, 9:14 PM
To: ECE 1111 <temple_engineering_ece1111@googlegroups.com>

The note below is something I want you to read and think about before you start ECE 1111. On top of learning how to code, we emphasize formatting, commenting, structure, reusability, etc. It isn't about just getting a working program, it is about writing good, clean, well-documented code. Keeping it simple is very important.

-Joe

-------- Forwarded Message --------
Subject: writing good software is like poetry... or a good song
Date: Sun, 18 Dec 2022 20:59:41 -0500
From: Joseph Picone <joseph.picone@gmail.com>
To: nedc_research <nedc_research@googlegroups.com>

I have often talked about how writing good software is like writing a good poem. A good poem sounds good, looks good, is simple, concise, and leaves you thinking "I couldn't have done it any better":

"The Old Pond" by Matsuo Bashō
An old silent pond
A frog jumps into the pond—
Splash! Silence again.

The same is true about a good song. I was listening to a few songs tonight from the 1960's and 70's that I have probably heard a thousand times. These songs are memorable for many reasons - some have a catchy melody, some have a phrase that makes the song become a cultural touchstone, and some just capture a moment in time that makes you remember a very specific event in history. It is often argued that a truly great song involves a combination of all of these things - the lyrics, the singing style and the melody must all come together for a song to make a lasting impression.

But what is amazing is you listen to the song and think how you wouldn't change one thing about it - it is just a perfect encapsulation of some thought, emotion or event (and I am not talking about novelty songs: https://digitaldreamdoor.com/pages/best_songs-novelty.html).

I find that writing good software gives you that same feeling. A useful program with an easy to use interface becomes part of your toolset and becomes second nature. You can't imagine life without that tool. I think an emacs macro is a good example of this - we could not have been nearly as productive without being able to use macros. I once saw a student spend a week writing a Java program to do something I did in emacs with a macro in less than a minute.

We have many examples of good code in the ISIP env. One of our oldest programs created at Temple is nedc_gen_feats - a very powerful tool that we enlisted to solve an issue with the Resnet decoder this past week. The specific features we exploited in that tool were written about 8 years ago or so and have not been used heavily until now. How is it that we had the foresight to solve a problem 8 years before needing that solution today?

Tools like gen_feats actually had a really cool graphical interface that let you prototype algorithms by drawing block diagrams:

https://isip.piconepress.com/publications/conference_proceedings/2000/dod_stw/asr/presentation/html/fe_02.html

It is sad we lost all this code over time because the GUI was quite popular.

There are many other examples, such as nedc_print_header (simple but powerful) and most recently the ResNet Seizure Detection system (a nice streamlined interface for what is a very complicated process).

Good, simple software doesn't happen overnight or easily. It takes a lot of time to think things through and arrive at a simple, clean interface. It takes long discussions with users, colleagues and programmers to achieve a good understanding of what the software needs to do.

If you are not inclined to thinking this way, then you are probably not "ISIP-material" when it comes to writing software. It is easy to write some complicated code that solves a very specific problem, but can't be reused for other tasks. It is common that a programmer will say "but I worked so hard on that." Complex programs end up taking a lot of time to maintain and they don't last over time. They are a horrible investment.

What really separates software developers in this group are the ones who strive for clean, simple, intuitive and easy to maintain interfaces. Your code should be a work of art 😉 and users should immediately be able to use it without extensive training.

That is much harder than it looks.

-Joe