Week 4 Journal

Rodger Byrd

I. Overview

For my area I'm looking at Anti-Patterns in code, these are also known as code smells. I've also seen them referred to as Atoms of Confusion and nano patterrns.

II. Journal Entry

For my learning process so far, I've looked at how to use google scholar to optimize my time looking for the latest research. Up until this class I would go onto the UCCS library website and use the research databases to find papers. It has been very helpful to set up my google scholar profile to automatically show the papers I can download through my UCCS credentails. I have a couple of things on my to-do list. I need to speak with Prof Boult about how to get a private github as a student, as that would be very useful, and I have a handful of questions I would like to get his opinion on. One thing I want to focus on is very recent relevant papers. Fortunately google scholar allows you to researc only papers that have been written since a particular year.

III. Journal 4

A. Who is the Main Character

The main character is the anti-pattern. I need to find a way to clearly define the anti-pattern conceptually.

Listing 1. Example Anti-Pattern

```
#DEFINE M 2+3

int main() {
  int x=5, y;
  y= x * M;
  cout << y << endl;
  return 0;
}
```

There are some authors who define it as interchangeable as a code smell. I don't think that is correct. I think code smells are precursors to anti-patterns. Meaning that an anti-pattern is a known bad problem, where a code smell may lead to a problem. An example anti-pattern is shown in listing 1. Some developers will expect the output to be 25 and some will be 13 because they don't understand intuitively how the macro function works (the correct answer is 13).

On second thought, is the character the developer? And is what is interesting about it their human nature which leads them to confusion? I would guess tha most developers think that a compiler is well-defined and bugs in code must be due to lack of understanding not code structure that is good at confusing human nature.

B. What Character Traits Make them Interesting

What's interesting about anti-patterns is that they have a very human aspect to them. They overlap the way humans think with the way code is written. They connect common ways of human misunderstanding to software development. Personally, I think most developers expect code to work as they understand it to. They don't spend a lot of time thinking about how code will work in ways they don't expect. It is in the nature of most engineers to see things in mathematical/binary ways and ignore the human aspects to what they are working on. Example boeing 737 max. Code developers expected pilots to respond with typical emergency procedure, but when the errors occurred they pilots were overwhelmed by the amount of feedback they recieved in the cockpit (need reference).

C. What do the Character Need to do or Get (Goal)

The anti-patterns need to be understood and identified. Once that happens developers can change their best practices. The best practices should be created in such a way that the typical misunderstandings of the developer

D. Why is That Goal Important (motive)

Note: not many papers about code smells talk about why it is important.

E. What Conflicts/Problems Block the Character

The problem is people don't realize they are implementing anti-patterns at the time they are writing code. The interesting things about it are how do we find them. How do we detect them, and what is the fix when we identify the problem.

F. How do they Create Risk and Danger

Because we know that the anti-patterns cause confusion in the developer, they create risk because they create unstable code. This is risk for the owner of the software and the customer of the developer who uses code

G. What Does the Character Do (Struggles) to Reach Goal

This is something I think is interesting, is how are these problems identified. It isnt' enought to just use expert advice/-knowledge. There must be a way to mathematically demonstrate or by experimentation that particular code patterns cause problems.

H. What Sensory Details Will Make the Story Seem Real

Real world examples of the problem

The files for this latex document are in the github repository located at `https://github.com/rodger79/CS6000`

Papers that were scanned and trashed are referenced in the bibliography below.

REFERENCES

[1] M. Mantyla, J. Vanhanen, and C. Lassenius, "A taxonomy and an initial empirical study of bad smells in code," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, Sep. 2003, pp. 381–384.

[2] R. Arcoverde, A. Garcia, and E. Figueiredo, "Understanding the Longevity of Code Smells: Preliminary Results of an Explanatory Survey," in *Proceedings of the 4th Workshop on Refactoring Tools*, ser. WRT '11. New York, NY, USA: ACM, 2011, pp. 33–36, event-place: Waikiki, Honolulu, HI, USA. [Online]. Available: http://doi.acm.org/10.1145/1984732.1984740

[3] A. Yamashita and L. Moonen, "Do developers care about code smells? An exploratory survey," Oct., pp. 242–251.

[4] V. Garousi and B. Kk, "Smells in software test code: A survey of knowledge in industry and academia," *Journal of Systems and Software*, vol. 138, pp. 52 – 81, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217303060

[5] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress in Informatics*, no. 5, p. 35, Mar. 2008. [Online]. Available: http://www.nii.ac.jp/pi/n5/5_35.html

[6] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158 – 173, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217303114

[7] D. Gopstein, J. Iannacone, Y. Yan, L. DeLong, Y. Zhuang, M. K.-C. Yeh, and J. Cappos, "Understanding Misunderstandings in Source Code," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, pp. 129–139, event-place: Paderborn, Germany. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106264

[8] S. Singh and S. Kaur, "A systematic literature review: Refactoring for disclosing code smells in object oriented software," *Ain Shams Engineering Journal*, vol. 9, no. 4, pp. 2129 – 2151, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2090447917300412