CS 311 - Introduction to Software Engineering

***Assignment 02: due (Friday, Oct 26, 2018)***

In the lectures, we discussed software architecture patterns and the characters of good architecture. However, in practice, it is common while developing a software system to drift towards a lousy architecture. The **Big Ball of Mud** Pattern is the most frequently deployed architecture in software systems. Read the paper "Big Ball of Mud" by Brian Foote and Joseph Yoder, then answer the following questions:

Q1) What are the main reasons that lead software developers to use bad architecture patterns such as the Big Ball of Mud? **[20 points]**

* Time: Not having enough time to produce a proper architecture.
* Experience: Lack of industry, domain, or architectural experience.
* Turnover: Losing team members and having to bring new programmers on mid or post production.
* Skill: The differing skills or lack of certain skills by programmers.
* Complexity: A complex problem may cause a complex solution.
* Change: Changes in the requirements or unexpected additions to the system.
* Cost: The high cost of putting the proper effort into architecture.

Q2) There are 6 poor software architecture patterns in the paper. Give a brief description of each pattern **[50 points]**

1. BIG BALL OF MUD

A haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle that shows unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated. The overall structure of the system may never have been well defined or has eroded beyond recognition.

1. THROWAWAY CODE

Quick-and-dirty code that was intended to be used only once and then discarded. However, such code often takes on a life of its own, despite casual structure and poor or non-existent documentation.

1. PIECEMEAL GROWTH

The relentless onslaught of changing requirements that any successful system attracts can gradually undermine its structure. Systems that were once tidy become overgrown as PIECEMEAL GROWTH gradually allows elements of the system to sprawl in an uncontrolled fashion. If such sprawl continues unabated, the structure of the system can become so badly compromised that it must be abandoned.

1. KEEP IT WORKING

Doing what it takes to maintain the software and keep it going. You avoid making too many or major changes to the current system with fear of new problems arising that cause the system to stop working. Instead new subsystems are created and used alongside and incorporated with the current system, as to keep the current system from going down since any new problems should be isolated to the new subsystem.

1. SWEEPING IT UNDER THE RUG

A simple way to begin to control decline is to cordon off the blighted areas and put an attractive façade around them. Essentially, we cover up or hide the poor or messy code similarly to sweeping dirt under the rug when cleaning your house. The poor or messy code is still there with its poor architecture, but at first glance it may appear to be less of a problem.

1. RECONSTRUCTION

The code has reached a point where it has become technically or economically obsolete. At this point a total rewrite of the code from the ground up is required. Sometimes new necessary changes may not be able to be incorporated in the current codebase, or the codebase is outdated, or the original programmers are all gone. Any of these may be call for a total rewrite of the system.

Q3) Select any two poor software architecture patterns from the six patterns listed in the paper and explain, how we could detect these patterns and how we can avoid them. Give clear examples to justify your answer. **[30 points]**

1. BIG BALL OF MUD
   1. Signs of this Pattern
      1. Data structures may be haphazardly constructed, or even next to non-existent.
      2. Every shred of important state data may be global.
      3. Variable and function names might be uninformative, or even misleading.
      4. Functions themselves may make extensive use of global variables, as well as long lists of poorly defined parameters.
      5. The functions themselves are lengthy and convoluted and perform several unrelated tasks.
      6. Code is duplicated.
      7. The flow of control is hard to understand, and difficult to follow.
      8. The programmer’s intent is next to impossible to discern.
      9. The code is simply unreadable, and borders on indecipherable.
      10. The code exhibits the unmistakable signs of patch after patch at the hands of multiple maintainers, each of whom barely understood the consequences of what he or she was doing.
      11. Missing documentation.
   2. Avoiding this Pattern
      1. Follow Kent Beck’s analysis of building software: Make it work. Make it right. Make it fast.
      2. Involve programmers with domain experience who can better predict the architectural needs of the system.
      3. Ensure that programmers have the proper tools and strong communication about the architectural goals.
      4. Learn to understand and navigate messy code, so that you can understand how to fix and improve it.
2. SWEEPING IT UNDER THE RUG
   1. Signs of this Pattern
      1. Look for poor or lacking exception handling, since the code may seem to be working with few random failures since obscure cases may have been ignored.
      2. More global information than necessary.
      3. Messy and convoluted code.
      4. Difficult to comprehend.
   2. Avoiding this Pattern
      1. Draw architectural boundaries between different parts of the system as you go or find loosely coupled sections and draw architectural boundaries there.
      2. Take global information and separate it into distinct data structures.
      3. Enforce communication between the separate data structures using well defined interfaces.
      4. Updating old interfaces or adding new interfaces to sections of run down code.
      5. Quarantine messy chunks of code to determine their functionality.