Lyndon Renaud
104 566 776
Software Engineering
Assignment 2

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

Architecture is often treated as a luxury. It frequently takes a backseat to other concerns such as cost, time-to-market, and programmer skills. Architecture can be seen as a risk that will consume resources better spent on other time constrained concerns.There are several main reasons why software developers use bad architecture patterns. These are:

Time

-architectural concerns must yield to more pragmatic ones as a project deadline approaches

Experience

-a lack of experience can limit the architectural sophistication brought to a system

Turnover:

-replacing people who work on a software system can lead to misunderstanding of old code or design

Skill

-programmers differ in their degrees of experience with particular domains, languages used, tool preference, and experience

Complexity

-software often reflects the inherent complexity of the application domain (essential complexity)
-organization of a system reflects the sprawl and history of the organization that built it and the compromises made along the way (Conway's Law)
-"site" boundaries. Problems arise when needs of an application force unrestrained communication across these boundaries.

Change

-architecture assumes that future changes will be confined to the design space encompassed by the architecture
-when these changes are not supported by the current architecture, it can be expensive or messy to change

Cost

-architecture is a long-term investment which does not pay off immediately
-architectural concerns may delay a products market entry, making long-term investments moot
-architectural investment has no value if the company is already bankrupt (due to delays in shipping the software)

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 big ball of mud is a haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle" [Foote & Yoder, 2]. It is code with no architectural structure. High coupling among other issues is a problem in these systems due to unregulated growth and repeated, expedient repair. The following 5 points are patterns that can lead to a big ball of mud.

2. THROWAWAY CODE

This refers to quick and dirt code which was meant to be used once and then discarded. It has casual structure and poor or non-existent documentation. Throwaway code is usually used for prototypes or as an alternative to reusing someone else's more complex code. When a deadline looms, a working but sloppy program can outweigh the unknown costs of learning someone else's library or framework. Time is frequently the decisive force that drives programmers to write throwaway code.

3. PIECEMEAL GROWTH

Constantly changing requirements can undermine a software's structure. Systems that were once well-defined and tidy become overgrown as "piecemeal growth" gradually allows elements of the system to sprawl in an uncontrolled fashion (similar to urban blight). When faced with a choice between building something elegant from the ground up, or undermining existing architecture to quickly address a problem, architecture usually loses. Piecemeal growth address forces that encourage change and growth, allowing opportunities for growth to be exploited locally. The biggest risk with piecemeal growth is that it will gradually erode the overall structure of a system and turn into a big ball of mud

4. KEEP IT WORKING

"Do what it takes to maintain the software and keep it going. Keep it working" [Foote & Yoder, 14]. A strength of this pattern is that modifications that break the system are rejected immediately. Selecting paths that do not undermine the system's viability avoids dead ends. This pattern emphasizes acute, local concerns at the expense of chronic, architectural ones. By taking small steps in any direction, it is never more than a few steps back to a working system.

5. SWEEPING IT UNDER THE RUG

When code is complicated and convoluted, it helps to isolate the messiness into one area. This keeps it out of sight and can set the stage for additional refactoring.

6. RECONSTRUCTION

Reconstruction is necessary when code has declined to the point where it is beyond repair or even comprehension. We throw away old code and start over. One reason for this might be that the previous system was written by people who are long gone. It is essential that a post-mortem review be done of the old system to see why it failed.

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]

Throwaway Code

Throwaway code is sloppy, quick-written code that is undocumented. Often a programmer will construct a minimally functional program, promising himself that a more elegant version will follow after. Sometimes, the elegant version is never made. Throwaway code should not be used as a permanent solution for a problem. It can be useful for understanding the domain of the problem and figuring out ways of solving it. Once a working solution is made as throwaway code, it should be rewritten properly as a more polished architecture results in a system that is easier to maintain and extend.

Piecemeal Growth

Piecemeal growth happens when a software requires changes that may undermine the current architecture. The architecture will erode with compromises that have to be made in modifications, improvements, and repairs. Piecemeal growth can be avoided by revisiting the architectural design and making changes to the design in order to accommodate changes that need to be made.