ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (65-67), Month: October - December 2015, Available at: www.researchpublish.com

Kludges, Their Relation with Anti-Pattern: Lava Flow and a Refactored Solution in Rapid Software Development

SACHIN B S

Vidyavardhaka College of Engineering

Abstract: **A study of relationship that shows how kludges cause anti pattern, in particular lava flow. The most vulnerable systems to this relationship which are small business software's that are developed using rapid software development. We provide a new refactored solution and a better development method for such systems.**

Keywords: **kludges cause anti pattern, in particular lava flow, small business software's.**

1. INTRODUCTION

A kludge in a software system program code is an inefficient, unfathomable fix to a problem that works nonetheless. To *kludge around* is to avoid a bug or some difficult condition by building a kludge, perhaps relying on properties of the bug itself to assure proper operation. It is somewhat similar to a workaround but in which case the programmer knows what is being done.

On the other hand an anti-pattern is a certain pattern in software development that is considered a bad programming practice.A lava flow is a anti pattern that is commonly found in systems that originated as research but ended up in production. It is characterized by the lava like "flows" of previous developmental versions strewn about the code landscape, which have now hardened into a basalt like, immovable, generally useless mass of code that no one can remember much, if anything, about.

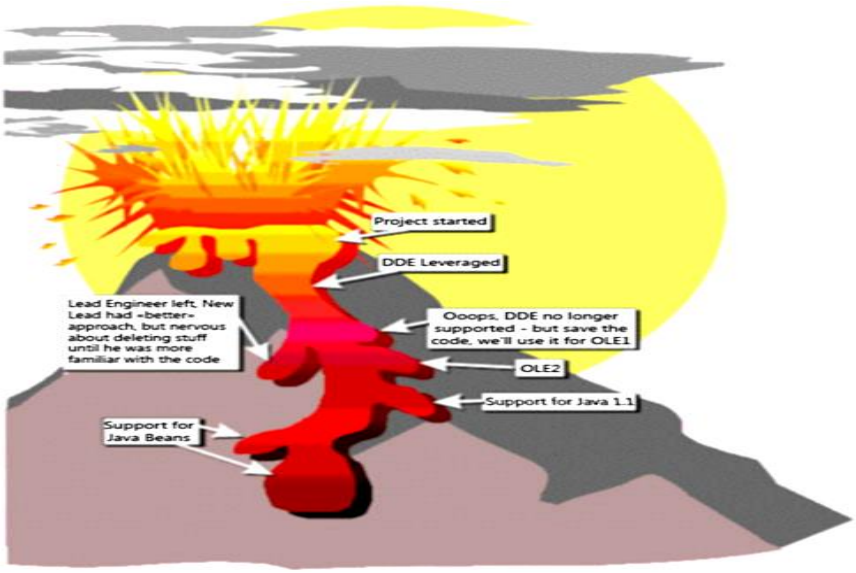

Fig.1.1 A typical lava flow anti pattern

This Anti-Pattern is incredibly common in innovative design shops where proof−of−concept or prototype code rapidly moves into production.

2. THE RELATIONSHIP

Kludges usually cause the program code to gradually degenerate into parts (specially in object oriented designs) that are in the code but whose use is unknown (undefined/poorly documented).This causes the typical lava flow pattern as shown fig.1.1.

Using mathematical notations we can say that if a code has 'n' workaround kludges then there must be at least 'm' globules of hardened code segments where n>m usually.

3. REFACTORED SOLUTION FOR THE RELATIONSHIP

3.1. Regular Solution for lava flow:

Ensure that sound architecture precedes production code development. This architecture must be backed up by a configuration management process that ensures architectural compliance and accommodates "mission creep" (changing requirements). If architectural consideration is short changed up front, ultimately, code is developed that is not a part of the target architecture, and is therefore redundant or dead. Over time, dead code becomes problematic for analysis, testing, and revision.

An important principle is to avoid architecture changes during active development. In particular, this applies to computational architecture, the software interfaces defining the systems integration solution. Management must postpone development until a clear architecture has been defined and disseminated to developers.

Tools such as the Source−Code Control System (SCCS) assist in configuration management. SCCS is bundled with most UNIX environments and provides a basic capability to record histories of updates to configuration controlled files.

3.2. New refactored Solution for small /medium scale projects:

Usually small business systems require rapid development of software which might encourage the introduction of kludges and hence Lava flow in iterative development.

In such systems we see rapid requirement changes in each iteration of a software making the whole system vulnerable to the relationship we have discussed so far. So, we propose a method of software development where both speed of development accuracy and sound architecture are not undermined. Note that, iterative development is not supported in this method. This involves the following steps:

- Generating scenarios: The system developers must interact with users often to generate scenarios of possible system use but a prototype is not developed until core system functionality and hence direction of development and system architecture are defined.

- Code breakdown: This involves following a top down approach to coding where peripherals are worked up first later looking into the core functionality. Though this might compromise larger testing for the core system the system architecture is developed better.

- Two way task division: One team of programmers might develop object codes while the other team can document it. This saves time and also provides a third person view to a someone who knows the direction of development but not the actual code. Thus, the documenting team can also review the objects developed thus eliminating possible creep of lava flow.

- Replacing kludges with workarounds: If the team must happen to use kludges it's use and reason for use must be known.

- Eliminating kludges: Once the code is documented the second team can work on eliminating kludges using Redudant code check tools and re-iterate the version of the developed software thus eliminating lava flow along.

4. CONCLUSION

The paper suggests an improved approach to eliminating the relationship and the components of poor architecture that hinders rapid software development by using a method that has an emphasis on honest code documentation and eliminating code redundancy this improving reusability and quality of code.

ACKNOWLEDGEMENTS

1. Prof. Naveen Kumar, Dept. Of IS&E, VVCE, Mysore.
2. Dr. Ravi Kumar V, HOD, Dept. Of IS&E, VVCE, Mysore.

REFERENCES

[1] Ian Somerville., Software engineering, "Rapid Software Development", vol 3, 2011, pg.273-289.

[2] Palmer and Felsing, 2002, vol 4, pg.7-93, pg.112-128

[3] Kludges.,en.wikipedia.org

[4] Wiley., Anti Patterns, Refactoring software, Architectue and project in crisis.

[5] B. Kitchenham et al "Systematic Literature Reviews in Software Engineering? A Systematic Literature Review", *Information and Software Technology*, vol. 51, p1, p7 -15 2009

[6] .A. Gehlert and A. Metzger *Quality Reference Model for SBA*, 2009

[7] *Jackson W. Granholm (February 1962).* How to Design a Kludge. *Datamation. Pp.30–31.*

[8] R. Kling and W. Scacchi, "Recurrent Dilemmas of Computer Use in Complex Organizations," *Proc. 1979 Nat',l Computer Conf.*, AFIPS Press, vol. 48, 1979, pp. 107-116.

Works Cited:

[9] Definitions of Kludge and Lava flow taken from Jackson *W. Granholm (February 1962).* How to Design a Kludge. *Data motion. pp. 30–31.*