Lava Flow

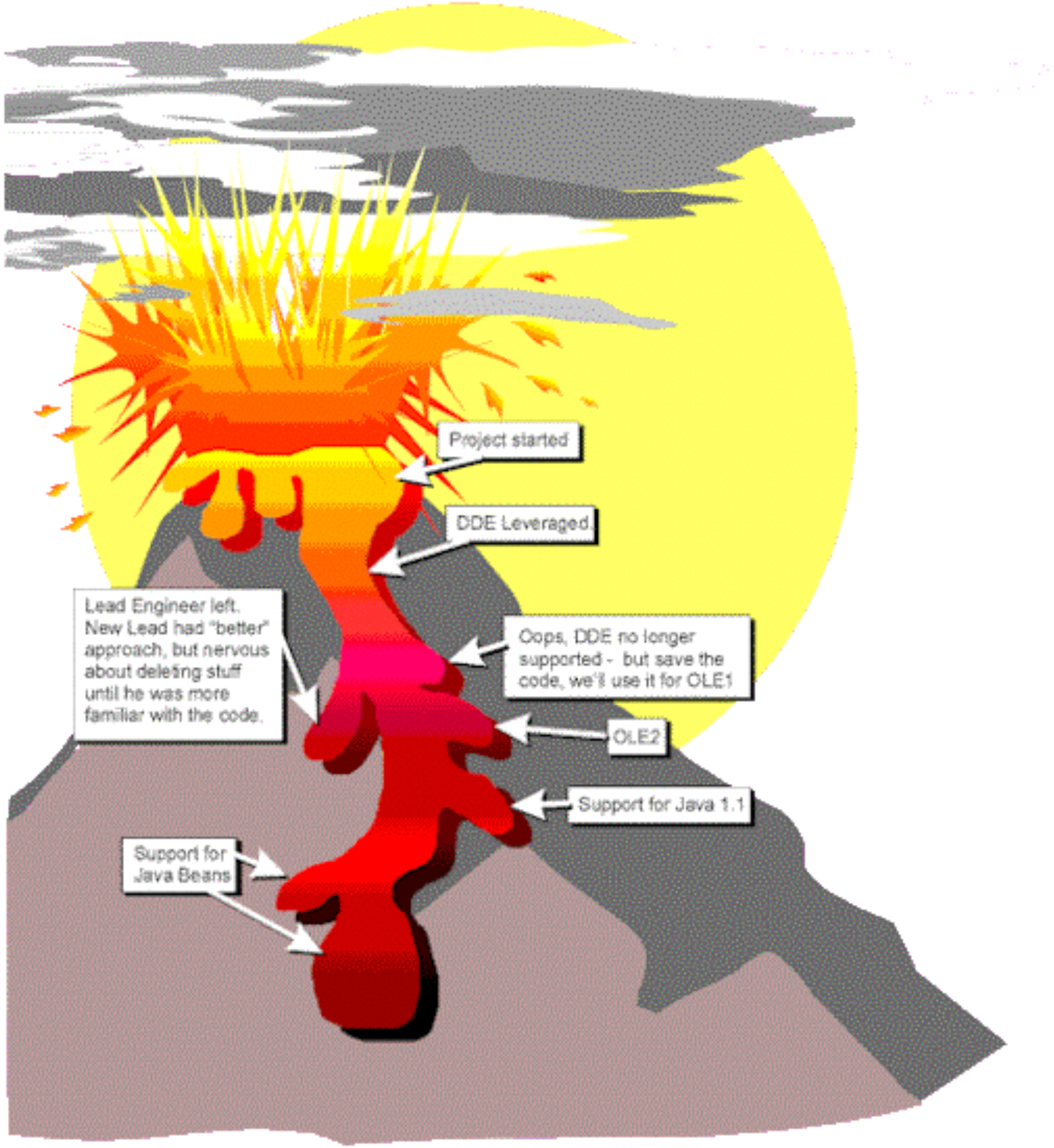
AntiPattern

21.10.2014

Jennifer Lück, Raffael Thome

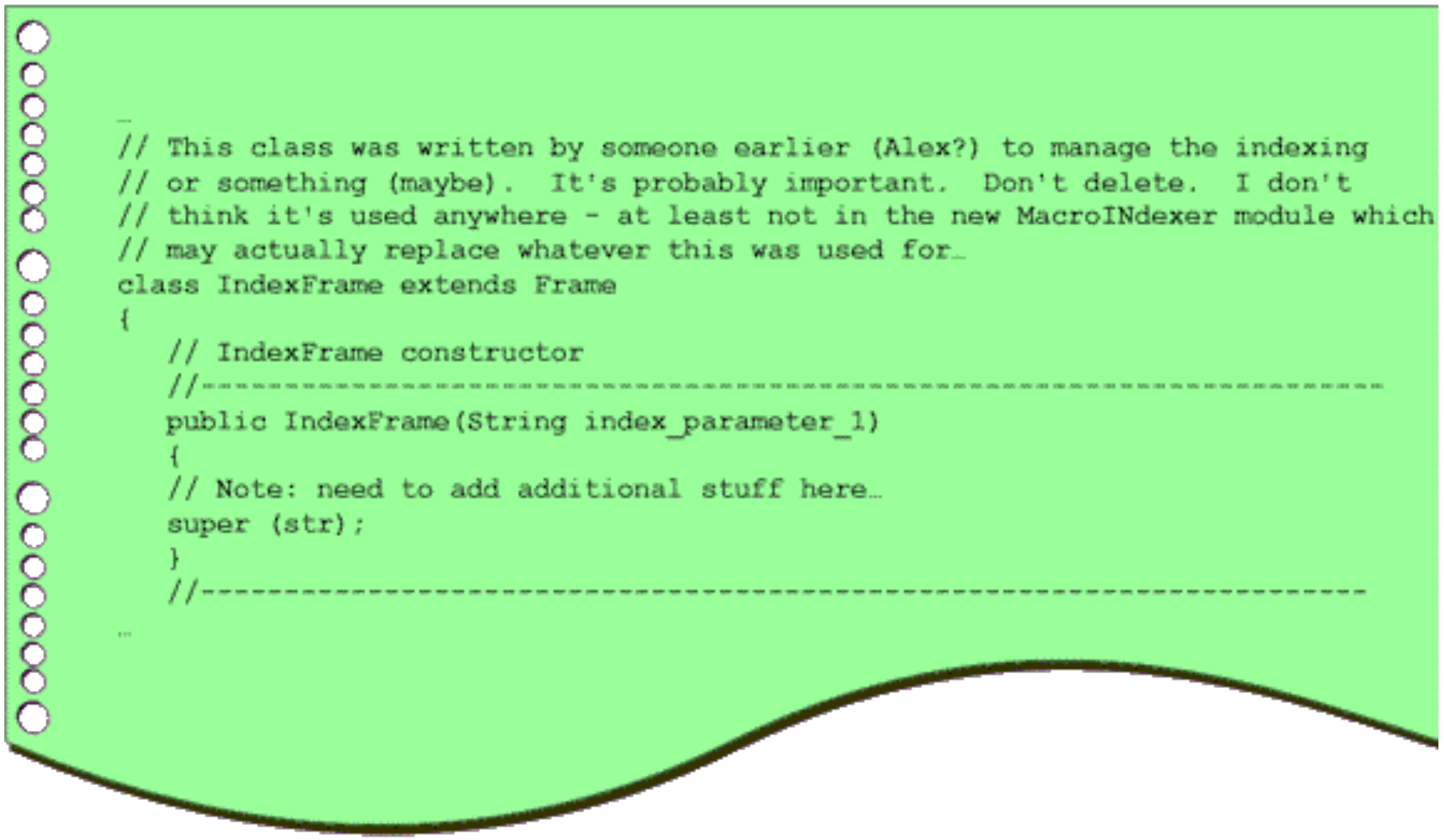
Anecdotal Evidence

"Oh that! Well Ray and Emil (they're no longer with the company) wrote that routine back when Jim (who left last month) was trying a work around for Irene's input processing code (she's in another department now, too). I don't think it's used anywhere now, but I'm not really sure. Irene didn't really document it very clearly, so we figured we just leave well enough alone for now… After all, the bloomin' thing works doesn't it?!"


Project started
DDE Leveraged.
Lead Engineer left. New Lead had "better" approach, but nervous about deleting stuff until he was more familiar with the code.
Oops, DDE no longer supported - but save the code, we'll use it for OLE1
OLE2
Support for Java 1.1
Support for Java Beans

Symptoms

- Unjustifiable code fragments and variables in the system
- Undocumented complex, important-looking functions, classes or segments which don't clearly relate to the system architecture
- Uncontrolled changes in system architecture
- Whole blocks of commented-out code
  - no explanation
  - no documentation
- Lots of "in flux" or "to be replaced" code areas
- Unused (dead) code
  - left in inexplicable or obsolete interfaces in header files



```
…
// This class was written by someone earlier (Alex?) to manage the indexing
// or something (maybe).  It's probably important.  Don't delete.  I don't
// think it's used anywhere - at least not in the new MacroINdexer module which
// may actually replace whatever this was used for…
class IndexFrame extends Frame
{
   // IndexFrame constructor
   //----------------------------------------------------------------------
   public IndexFrame(String index_parameter_1)
   {
   // Note: need to add additional stuff here…
   super (str);
   }
   //----------------------------------------------------------------------
…
```

Consequences

- If existing lava flow code is not removed, it can get multiplied by reusing in other areas
- If the process that leads to lava flow is not checked, there can be an exponential growth by producing new secondary flows while trying to “work around” the original ones
- As the flows compound and harden, it rapidly becomes impossible to document the code or understand its architecture enough to make improvements

Typical Causes

- Prototype code placed into production without thought toward Configuration Management
  - Uncontrolled distribution of unfinished code
  - Implementation of several trial approaches toward implementing some functionality
- Single developer (lone wolf) written code
- Lack of architecture, or non-architecture driven development
  - Especially prevalent with highly transient development teams
- Iterative Development Process
- Architectural Scars

Preventing Lava Flow

- Ensuring that solid architecture precedes production code development - backed up by a CM

- Avoid architecture changes during active development

- Establish stable system-level software interfaces

- Customise a quasi CM process during research that can readily be scaled into a full-blown CM control system

Solutions

- Postpone development until a clear architecture has been defined and communicated to developers
- System discovery is required to
  - locate the components that are really used and necessary to the system
  - identify those lines of code that can be safely deleted
- If the suspected dead code is eliminated and bugs occurred
  - Resist the urge to immediately fix the symptoms without fully understanding the cause of the error
  - Study dependencies

AntiPattern Template

AntiPattern Name:	Lava Flow
Also Known As:	Dead Code
Most Frequent Scale:	Application
Refactored Solution Name:	Architectural Configuration Management
Refactored Solution Type:	Process
Root Causes:	Avarice, Greed, Sloth
Unbalanced Forces:	Management of Functionality, Performance, Complexity

Sources

- www.antipatterns.com/lavaflow.htm
- http://en.wikipedia.org/wiki/Lava_flow_(programming)