

Vorstellung der Antipattern Lava-Flow und Poltergeists

Von Claas Reim

Inhaltsverzeichnis

1. Lava Flow:

- 1.1. Einführung
- 1.2. Ausnahmefall
- 1.3. Lösungsvorschläge
- 1.4. Beispielfall

2. Poltergeists:

- 1.1. Einführung
- 1.2. Lösungsvorschläge
- 1.3. Beispielfall

1. Lava Flow

1.1. Einführung

- auch bekannt als:
 - ungenutzter Programmcode (dead code)
- allgemeine Form:
 - Forschungsprogramme, die nun in der Produktion eingesetzt werden sollen
 - Quelltextfragmente von früheren Testläufen oder Programmpräsentationen
 - um solche Passagen wird meist herumgearbeitet

1.1. Einführung

The diagram shows a volcano with a lava flow. The flow starts at the top and moves downwards, branching out. Several callout boxes point to different stages of the flow:

- Project started**: Points to the top of the volcano.
- DDE Leveraged**: Points to the upper part of the lava flow.
- Lead Engineer left. New Lead had "better" approach, but nervous about deleting stuff until he was more familiar with the code.**: Points to a section of the flow.
- Oops, DDE no longer supported - but save the code, we'll use it for OLE1**: Points to a section of the flow.
- OLE2**: Points to a section of the flow.
- Support for Java 1.1**: Points to a section of the flow.
- Support for JabaBeans**: Points to the bottom of the flow.

On the right side, a green box contains a code snippet:

```
0 ...  
1 // This class was written by someone earlier (Alex?) to manage the  
2 // indexing or something (maybe). It's probably important.  
3 //  
4 // ****DO NOT DELETE!****  
5 //  
6 // I don't think this is used anywhere - at least not in the new  
7 // Macro_Indexer module which may actually replace whatever  
8 // this was used for, but I'm not sure, so it's best to just leave it  
9 // here for now... (J.P. - 4/89)  
10  
11 class IndexFrame extends Frame  
12 {  
13     // IndexFrame constructor  
14     //-----  
15     public IndexFrame(String index_parameter_1)  
16     {  
17         // Note: need to add additional stuff here...  
18         super (str);  
19     }  
20     //-----  
21 }
```

1.1. Einführung

- Gründe für die schlechten Entwurfseigenschaften des Lava Flusses
 - sind sehr kostenaufwendig zu lokalisieren
 - Verschwendung von Ressourcen und Rechenleistung
 - Verlust von Vorteilen der objekt-orientierten Programmierung

1.1. Einführung

- Symptome:
 - ungenutzte Variablen und Quelltextfragmente
 - undokumentierte Abschnitte; wichtig aussehende Funktionen und Klassen
 - längere auskommentierte Quelltextpassagen ohne Kommentierungen
 - mehrere duplizierte Abschnitte
 - veraltete Schnittstellen in Header-Datei

1.1. Einführung

- Konsequenzen:
 - wieder verwendeter Quelltext in anderen Klassen
 - Entstehung von Redundanzen beim Überarbeiten von existierenden Lava Flüssen
 - Programme sind schwierig zu dokumentieren und strukturell zu verstehen
 - spätere Verbesserungen nur mit hohem Aufwand machbar

1.1. Einführung

- Ursache:
 - Forschungsprogramme werden in Produktionsanwendungen umgeschrieben
 - unkontrollierte Verteilung von nicht fertigem Quelltext
 - schlechte Konfigurationsverwaltungen
 - mangelnde Strukturierung
 - sich wiederholende Entwicklungsabläufe

1.2. Ausnahmefall

- Bsp.: kurzfristige Präsentation der Architektur eines Systems
- wichtige Komponenten kennzeichnen
- überflüssigen Code verbleibt im Quelltext

1.3. Lösungsvorschläge

- Verwaltungssystem zum Erkennen und Entfernen von „totem Programmcode“
- Architektur des Systems muss mit dem Quelltext weiterentwickelt werden
- stabile und wohl definierte Schnittstellen implementieren
- vorsichtiges Entfernen von ungenutztem Quelltext

1.4. Beispielfall

- Datamining anhand eines vorhandenen Systems
- Auftreten von großen Segmenten von Quelltext, der keinen Bezug zur Architektur des Systems hatte
- Zitat eines Entwickler beim Nachfragen zu den Quelltextfragmenten:
„Ach das! ... Reggie hatte irgendwas probiert, ... Ich glaube ein Teil von Reggies anderem Code hängt von diesem ab, so dass wir diesen nicht gelöscht haben.“

1.4. Beispielfall

- Vorstellung des System sollte kurzfristig vorbereitet werden
- eigenständiges Erschließen der Architektur anhand des Quelltextes war nicht möglich
- mit Hilfe eines Entwicklers wurden die Schnittstellen in IDL formuliert
- Erstellung der Produktpräsentation des Datamining über die IDL mit Hilfe einer kleinen Corba-Anwendung

2. Poltergeists

2.1. Einführung

- auch bekannt als:
 - Zigeuner (-wagen),
 - Vermehrung von Klassen



2.1. Einführung

- Allgemeine Form:
 - Klassen mit beschränkter Verantwortung und kurzer Lebensdauer
 - initialisieren meist Vorgänge in langlebigen Klassen
 - werden meist von unerfahrenen Entwicklern verwendet

2.1. Einführung

- Gründe für die schlechten Entwurfseigenschaften des Poltergeistes
 - Verschwendung von Ressourcen
 - Ineffizienz
 - Behinderung eines objekt-orientierten Entwurfs

2.1. Einführung

- Symptome:
 - redundante Navigationspfade
 - kurzlebige Verbindungen und Klassen
 - zustandslose Klassen
 - Klassen, die lediglich andere Klassen über eine Operation aufrufen oder initialisieren
 - offensichtliche Klassennamen mit Suffixen wie „controller_“ oder „manager_“

2.1. Einführung

- Konsequenzen:
 - unnötige Klassen und Komplexität stören das Verständnis
 - schlechte Wartbarkeit
 - Änderbarkeit

2.1. Einführung

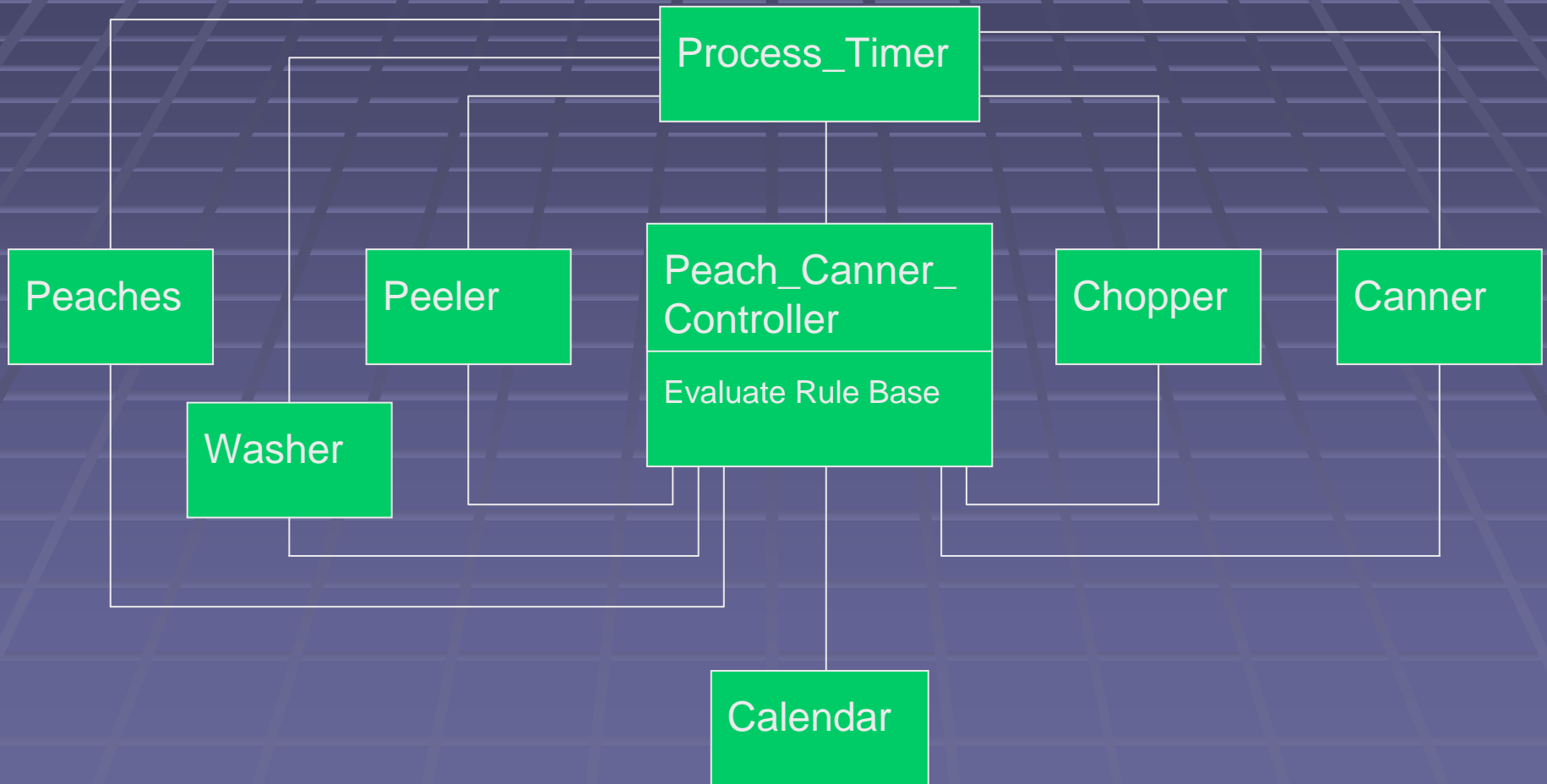
- Ursache:
 - Fehlen von einer objekt-orientierten Struktur
 - falsches Werkzeug (OO) für die Aufgabe

Zitat: „Es gibt keinen richtigen Weg, um das falsche zu tun.“

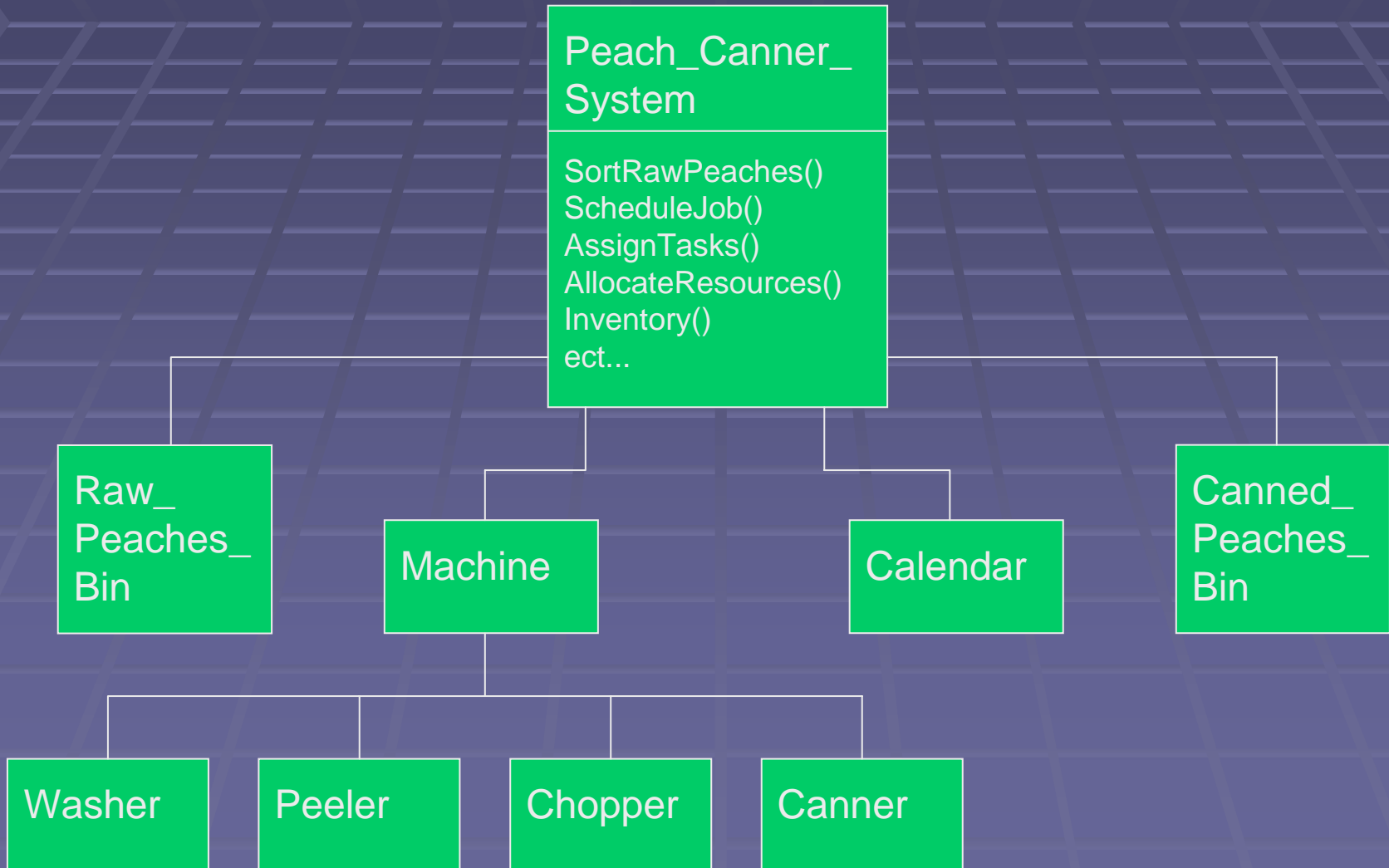
2.2. Lösungsvorschläge

- irrelevante Klassen eliminieren
- kleinere Aufgaben zu kohärenten größeren Klassen zusammenfassen
- Funktionalität der „Poltergeist“-Klassen muss wieder hergestellt werden

2.3. Beispielfall



2.3. Beispielfall



**Vielen Dank für Ihre
Aufmerksamkeit**

Fragen ???