Jay Patel

Christian Klein

Joseph Thill

Derek Yu

Zifeng Jiang

Studio 11: Design Strategies

# Architecture

Beaker is a web application that best matches with the Server/Client architecture. The front end implements the MVC structure with Javascript while the Java backend utilizes a collection of servers and NGINX for scalability. The front end relays some user input to the server in order to be handled appropriately.

# Design Patterns

|  |  |  |
| --- | --- | --- |
| **Name** | **Location** | **Description** |
| Builder | Does not exist within Beaker | Creation of objects doesn’t require a granular design pattern like builder to create them. Beaker mainly relies on factories to create objects. |
| Factory Method | plugin\sqlsh\src\main\java\com\twosigma\beaker\sqlsh\autocomplete\db | DbExplorerFactory.java is a factory class. One concrete class, DbExplorer, extends this factory class. |
| Prototype | \core\src\main\web\app\mainapp  Line 1229 | Keyword “.this” is used for prototype design pattern. For example if we need to use an object for this pattern it will clone the previously created object. In Beaker they use “.this” in many places where they are specifically calling the one that was declared in the constructor. |
| Singleton | /[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/beaker.js  line 45 | getObjectMapper creates and returns only one and the same instance of ObjectMapper. |
| Singleton | /[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/beaker.js  Entire class | When running Beaker, you have exactly one instance of the beaker class that is the root of everything and sets up server routing. |
| Abstract Factory | plugin/jvm/src/main/java/com/  twosigma/beaker/jvm/updater  Three separate classes | There are a series of Factory Java classes to create various Updater classes. The UpdaterFactory class (technically interface) is the abstract factory class used to create objects extended from the Updater abstract class. The concrete classes that extend these classes are ObservableUpdaterFactory and BeakerDashboardUpdaterFactory. These each make their obvious Updater objects. |
| Adapter | Does not exist within Beaker | All the inputs in Beaker are codes and designed functions. There is no need to import extra Adapter to read these inputs. |
| Bridge | Does not exist within Beaker | There is no need to use this design pattern since it is used to decouple an abstraction from its implementation so that the two can vary independently. |
| Composite | Does not exist within Beaker | Beaker doesn’t use the composite pattern to represent trees or hierarchies within the program. |
| Decorator | Does not exist within Beaker | N / A |
| Facade | Does not exist within Beaker | N / A |
| Flyweight | /core/twosigma/beaker/core/module/config/ | There are four classes in this folder containing many shared data and settings. |
| Proxy | \core\src\main\web\app\mainapp | There is a wrapper that helps the method that has been called which provides for another object to access it, also helps to support and control the access of the objects that have been called within the method. |
| Chain of Responsibility | shared\src\main\java\com\twosigma\beaker\shared\module\util\GeneralUtilsImpl.java | GeneralUtilsImpl.java has a set of file system methods which handle four different file formats. Depending on the file format, the responsibility gets passed toward the correct method. For example, there are four readFile methods which take a Path, File, String, and URI respectively. |
| Command | /core/twosigma/beaker/core/module/elfinder/impl/commands | There are plenty of command classes in this folder. All of them implement Command class and represent different functions. However, each of the functions is packaged in one specific classes to apply Command design pattern. |
| Interpreter | \plugin\cpp\src\main\java\com\twosigma\beaker\cpp\utils\Extractor.java | Extractor extends the base class CPP14BaseListener and expands on grammar parsing logic. |
| Iterator | Does not exist within Beaker | Beaker has collections that are iterated through, but there are no custom iterator classes. |
| Mediator | core/src/main/web/app/helpers/helper.js  Line 33 | bkHelper is defined as a mediator between each plugin language and the rest of the Beaker program. Each plugin interacts only with the bkHelper object, rather than with each other or with Beaker directly. |
| Memento | [beaker-notebook](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb)/[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/[mainapp](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp)/[components](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp/components)/[notebook](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp/components/notebook)/cell-directive.js  Line 89 | Beaker saves the last created cell’s language and sets that as the default cell language for new cells. |
| Observer | \core\src\main\web\app\mainapp  Line 1127 | This is an event handler named keydownHandler. It observes/listens for key events and reacts differently when distinct key combinations are pressed. |
| State | [beaker-notebook](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb)/[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/[mainapp](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp)/[components](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp/components)/[notebook](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp/components/notebook)/cell-directive.js  Line 76 | Beaker cells have many states, one of which is the locked state. When a cell is locked, certain actions cannot be performed. The cell’s language is also saved as a state. |
| Strategy | \core\src\main\web\app\mainapp | There are some methods that have been used in this file that abstracts in an interface that changes the guts of an object. |
| Template Method | Does not exist within Beaker | N / A |
| Visitor | Does not exist within Beaker | N / A |
| Singleton | /[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/beaker.js  line 51 | getConnector creates and returns only one and the same instance of Connector. |
| Factory Pattern | /[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/beaker.js  OutputDisplayFactory, OutputDisplay, bkOutputDisplay | These classes follow the Factory design pattern because within the code for the OutputDisplay, there are attributes for the many plugins and features in the UI of Beaker. |

# Anti Patterns

## The Blob

The blob occurs when a majority of responsibilities is contained within a single, “god” class. This class contains a bulk of attributes and/or methods that other classes need to function. This forces these other classes to have multiple dependencies with the blob, transforming them into simple, redundant, data classes.

No, Beaker is not a victim of The Blob.

This anti pattern has been avoided within Beaker by dividing key functionality into layers. The developers split the layers into views, service, and backend which demonstrates separation of concerns.

## Lava Flow

Lava Flow is the existence of old, poorly documented code that often appears spaghetti-like, but no one on the development team is quite sure what it does or how to deal with it. This typically occurs on products that began as research or proof-of-concepts that eventually became full projects.

No, Beaker is not a victim of Lava Flow.

The majority of Beaker’s code is well documented and modular. Since the version of Beaker being examined is still in-progress, there may not have been much opportunity for Lava Flow to even begin. Let’s hope it stays free of Lava Flow as the project continues.

## Functional Decomposition

Functional decomposition occurs when classes are designed and related in terms of functions and subroutines instead of class hierarchies and their relationships. Instead of having a class have both attributes and several operations to interact with other classes in the system, a class would simply have one or few functions that it runs with entirely private members. It doesn’t use the advantages of an object-oriented design and only runs a particular subroutine.

No, Beaker has no occurrences of Functional Decomposition.

All classes are named and defined as objects with specific purposes, such as Serializer, Updater, and their subclasses. It seems like they were careful when defining classes to ensure that each class not only contributes to the class hierarchy, but also always has both attributes and operations that serve its particular function or role in the program. No class seems able to be simply moved as a function into another class, which is a good sign that Functional Decomposition isn’t occurring.

## Poltergeists

Poltergeists are classes within a system that have little to no functionality and an extremely short lifespan. These classes offer miniscule purpose and often clutter the software system or attempt to over-abstract things, thus creating a less-than-ideal product. This usually occurs when a developer is new to object-oriented programming.

Yes, poltergeists are found within Beaker.

Since the Beaker version we are looking at is in-progress, it’s tough to say whether or not these “poltergeists” are due to poor programming or are placeholders for future code, but an example poltergeist in the current code is pluginmanager.js located at [beaker-notebook](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb)/[core](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core)/[src](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src)/[main](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main)/[web](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web)/[app](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app)/ [mainapp](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp)/[components](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp/components)/[pluginmanager](https://github.com/twosigma/beaker-notebook/tree/e8427bf770a39aa262e4d166a760485b57e73dcb/core/src/main/web/app/mainapp/components/pluginmanager)/pluginmanager.js. This class is empty at the moment and has no methods.

## Golden Hammer

Golden Hammer means that the developer uses inappropriate and inefficient methods to solve problem. Maybe the developer is really familiar with one specific pattern so that he can apply it on many different programs. However, the inappropriate and inefficient method increases the potential risk of these programs. Which means that choosing the proper tool to solve corresponding problems is important during development process.

No, Golden Hammer doesn’t appear in Beaker.

In fact, as the popular object-oriented programming pattern, the MVC structure is appropriate to be the main pattern of Beaker. This pattern leads every classes to be purposive and efficient. So the developer can use proper pattern in every part of Beaker. .

## Spaghetti Code

Spaghetti Code means that one small method includes a complex control structure. For instance, the control structure contains many “if” and “goto”. This complex pattern makes the code hard to understand and edit. It doesn’t separate one big function into many small components successfully.

No, Beaker doesn’t include Spaghetti Code.

Spaghetti Code is an usual mistake for beginner. But it is easy to avoid in any proven program. The developers just make sure that not solving one complete puzzle in only and not citing too many methods in small structure.

## Cut-and-past Programming

Cut-and-paste programming usually means that there is a chunk of code that is being copied and pasted at bunch of places in order to do some functionality. But this does not mean that your code is effective because if there is an error in a code that you copied from somewhere and pasted in your program you will still have that error in your code which then later results into difficult to fix code of a particular mistake.

No, Cut-and-Paste Programming is not found in Beaker.

This anti pattern has been avoided within Beaker by calling the function instead of copying and pasting same lines of code.

**Com S 362 — Object-Oriented Analysis and Design**

**Certification of Individual Contribution and Understanding Form**

Project Name: Design Strategies Study Report Team Facilitator: Derek Yu

Homework Set: Studio #11 Date: 11-27-2016

Directions: Enter each team member’s name, and a “work rating” (1 - 5 with 5 as high and 1 as low score) that corresponds to the relative share of work done by the team member. Comments to explain work ratings other than 3 must also be provided. The total of the ratings must add up to 3 times the number of team members. Each team member must sign at the end of the form showing your agreement to your team member’s contribution.

|  |  |  |  |
| --- | --- | --- | --- |
| Printed team member’s name | Work  Rating  (1 - 5) | Comments/Explanations | Understanding  Self Rating  (0 to 100%) |
| Christian Klein | 3 | Architecture  Design Patterns  Anti Patterns | 90% |
| Derek Yu | 3 | Architecture  Design Patterns  Anti Patterns | 90% |
| Jay Patel | 3 | Design Patterns  Anti Patterns | 90% |
| Joe Thill | 3 | Design Patterns  Anti Patterns | 90% |
| Zifeng Jiang | 3 | Design Patterns  Anti Patterns | 90% |
| Total: | 15 |  | |

Note that you must make Total = 3 \* (Number of team members)

|  |  |
| --- | --- |
| I agree to the above ratings and understand our team’s solution. | |
| Christian Klein | Joe Thill |
| Derek Yu | Zifeng Jiang |
| Jay Patel | |