

# GLACIAL REFACTORING

A Glacier-inspired Approach to Code Cleanup



ROSE HOOPER | PYOHIO 2023

# WHY GLACIERS?

# WHY GLACIERS?

**GLACIAL REFACTORING IS A GRADUAL PROCESS.**

**Several distinct stages**

We can draw parallels between code features and glacial features.

# GLACIAL REFACTORIZING STAGES

- Codescape Accumulation
- Refactoring Phase
- Evolutionary Codescaping
- Accumulation (Pre-Glacial Period)
- Glacial Advance
- Glacial Retreat

# CODESCAPE

All software has a shape. I call this a **codescape**.

Some code is easy to follow, some is rugged and complex.

# CODESCAPE ACCUMULATION

## The beautiful beginnings.

Initial concept. Startup/project formed. Prototype. Adoption.

Features added, capabilities changed.

New developers, more contributors, more ideas. More capabilities.

Plugins, microservices, 3rd party APIs.

# THE EVER EVOLVING ECOSYSTEM AROUND US

Every-day influences shape the codescape.

Bigger. Better. Faster.

New hardware  new frameworks  new languages  new  
tools .

**NEW EVERYTHING!**

**NEW EVERYTHING!**

**EVERYTHING IS AMAZING!**



## BUT WAIT

The promised land of bug-free, performant, easy to modify, unit-tested code **doesn't exist.**

The unit tests are fragile.

Some integration tests don't run.

There are magics sprinkled in the code and deployment toolchain.

Even cosmic rays cause problems.

Software projects have an evolving codescape.

# GRADUAL ACCUMULATION OF CRUFT AND CODE FEATURES

Well-intentioned refactorings. ~~Aborted~~ features.

Backwards compatibility. Database cruft. Relics. Business priorities  
**Team churn.** and community pressure.

Rushed code. Misunderstood code. Magic dev environments.  
Ownership changes. Outsourcing. **Unsupported EOL**  
**versions of things;** Friendly and **hard forks.** Broken dependencies.  
**Ubuntu 12; Python 2.7, 3.4; RHEL/CentOS 5.**

~~Unmaintained dependencies. Performance fixes.~~

Cloudification. Scope creep, deadlines, break-fix. Community issues,  
disagreements.

My head hurts.

# GLACIAL REFACTORING

- Gradual process intended to take time.
- Frequent small changes with minimal scope.
- Safe, easy to roll-back changes.
- Self-documenting next steps.
- Easy to incorporate into any release lifecycle.

# GLACIAL REFACTORING

## ***THE FIRST REFACTORING PHASE***

During this phase, we'll act like glacial detectives, looking for clues in the structure of our code.

Our goal? To gradually flatten and streamline the code, smoothing out those rugged 'code mountains' we encounter.

## IDENTIFYING THE TERRAIN

The first step in glacial refactoring is understanding our terrain. We'll look for and identify features in our codescape that have emerged over time.

## EXAMPLE CODESCAPE FEATURES

- **Feature Moraines:** Accumulated layers of code, diverse and large, each layer a testament to our evolving project.
- **Code Erratics:** Isolated pieces of code, remnants of past needs, adrift in our current codescape.
- **Workaround Eskers:** Trails of quick fixes and shortcuts, weaving through our code, often born from urgent needs.
- **Deadline Scars:** Marks of rushed development; hasty patches, unpolished code, the product of tight timelines.
- **Data Kettles:** Pockets of data, scattered and disconnected, leftovers from the flow of information over time.
- **Code Mountains:** Deeply nested and complex code structures, resembling the rugged terrain carved by glaciers.



# SOME EXAMPLE CODESCAPE FEATURES

## EXPLORING FEATURE MORAINES IN CODE

Let's examine a code sample illustrating potential Feature Moraines  
— areas where code has accumulated over time.

```
1 from db.manager import db_connection, dsdb
2
3 class GlacierAnalysisTool:
4     def __init__(self, db=db_connection): ...
5     def analyze_ice_thickness(self): ...
6     def map_glacial_retreat(self): ...
7
8 class MoraineDataProcessor(GlacierAnalysisTool):
9     def aggregate_debris_data(self): ...
10    def calculate_moraine_age(self): ...
11    def import_initial_core_analysis(self, path): ...
12    def legacy_data_cleanup(self): ...
13    def add_core_analysis(self, core_id, url=CORE_API, provider="USGS"): ...
```

Hints of legacy data and and importing initial data.

# TRACING THE SYSTEM'S ARCHITECTURE

Here, we find hints of our system's architecture, like a separate Data Science database. This knowledge helps guide our refactoring strategy.

```
1 from db.manager import db_connection, dsdb
2
3 class GlacierAnalysisTool:
4     def __init__(self, db=dsdb): ...
5     def analyze_ice_thickness(self): ...
6     def map_glacial_retreat(self): ...
7
8 class MoraineDataProcessor(GlacierAnalysisTool):
9     def aggregate_debris_data(self): ...
10    def calculate_moraine_age(self): ...
11    def import_initial_core_analysis(self, path): ...
12    def legacy_data_cleanup(self): ...
13    def add_core_analysis(self, core_id, url=CORE_API, provider="USGS"): ...
```

# DOCUMENTING WITH GLACIAL NOTES

As we refactor, it's vital to add 'Glacial Notes' — comments that document our observations and action items. This helps both our understanding and that of future developers.

```
1 from db.manager import db_connection, dsdb
2 # GlacialNote: dsdb appears to be a data science database that we don't have access to.
3
4 class GlacierAnalysisTool:
5     def __init__(self, db=dsdb): ...
6     def analyze_ice_thickness(self): ...
7     def map_glacial_retreat(self): ...
8
9 class MoraineDataProcessor(GlacierAnalysisTool):
10     __glacial_note__ = "Flatten: Pull base up, this is the only use of GlacierAnalysisTool."
11     def aggregate_debris_data(self): ...
12     def calculate_moraine_age(self): ...
13     def import_initial_core_analysis(self, path):
14         # GlacialNote: Appears to be unused, no refs in current code
```

## **FLATTENING CODE MOUNTAINS**

A 'Code Mountain' is an unusually deeply nested code formation. One of the most intimidating naturally occurring code formations to work with. Usually full of Code Moraines and Code Erratics.

# FLATTENING CODE MOUNTAINS

The distinct shape of a Code Mountain is easy to spot. The left margin is shaped like the peaks and valleys of a mountain range.

```
1 | for slide in presentation:
2 |     |if slide.is_have_content():
3 |         |for elemento in slide.elements:
4 |             |if elemento.is_textual():
5 |                 |for parrafo in elemento.parrafos:
6 |                     |if parrafo.is_have_special_format():
7 |                         |if parrafo.is_bold():
8 |                             elemento.apply_bold(parrafo)
9 |                         elif parrafo.is_italic():
10 |                             elemento.apply_italic(parrafo)
11 |                         |else:
12 |                             |if sys.getenv("FF_20090101_0PS-2319"):
13 |                                 sys.exit("Unsupported format - 不适合")
14 |                             continue
15 |                         |else:
16 |                             parrafo.normalize_format()
17 |             |else:
18 |                 |for img in elemento.media:
19 |                     |if img.is_fit(slide):
20 |                         |if media_utils.needs_light_adjustment(img):
21 |                             slide.add_img(img, bg=(255,255,255))
22 |                         |else:
23 |                             raise NotImplementedError('फिट नहीं होता')
```

# FLATTENING CODE MOUNTAINS

Here we add a couple Glacial Notes documenting findings before changing code.

```
1 for slide in presentation:
2     if slide.is_have_content():
3         for elemento in slide.elements:
4             if elemento.is_textual():
5                 for parrafo in elemento.parrafos:
6                     if parrafo.is_have_special_format():
7                         if parrafo.is_bold():
8                             elemento.apply_bold(parrafo)
9                         elif parrafo.is_italic():
10                            elemento.apply_italic(parrafo)
11                    else:
12                        # GlacialNote: Flag no longer set, JIRA issue closed in 2022
13                        # Clue: merge commit for REL-4832 fixed in presentation UI.
14                        # Action: Remove
15                        if sys.getenv("FF_20090101_OPS-2319"):
16                            sys.exit("Unsupported format - 不适合")
17                    continue
18                else:
19                    parrafo.normalize_format()
20            else:
21                # GlacialNote: Good spot to flatten or extract to functions
22                for img in elemento.media:
```

**COMMIT**



# HAULING AWAY CODE ERRATICS

With our intent committed, we can now make fixes, one at a time.  
One example is Code Erratics..

```
1 for slide in presentation:
2     if slide.is_have_content():
3         for elemento in slide.elements:
4             if elemento.is_textual():
5                 for parrafo in elemento.parrafos:
6                     if parrafo.is_have_special_format():
7                         if parrafo.is_bold():
8                             elemento.apply_bold(parrafo)
9                         elif parrafo.is_italic():
10                            elemento.apply_italic(parrafo)
11                    else:
12                        # GlacialNote: Flag no longer set, JIRA issue closed in 2022
13                        # Clue: merge commit for REL-4832 fixed in presentation UI.
14                        # Action: Remove
15                        if sys.getenv("FF_20090101_OPS-2319"):
16                            sys.exit("Unsupported format - 不适合")
17                    continue
18                else:
19                    parrafo.normalize_format()
20            else:
21                # GlacialNote: Good spot to flatten or extract to functions
22                for img in elemento.media:
23                    if img.is_fit(slide):
```

# HAULING AWAY CODE ERRATICS

Add Glacial Notes as needed.

```
1 for slide in presentation:
2     if slide.is_have_content():
3         for elemento in slide.elements:
4             if elemento.is_textual():
5                 for parrafo in elemento.parrafos:
6                     if parrafo.is_have_special_format():
7                         if parrafo.is_bold():
8                             elemento.apply_bold(parrafo)
9                         elif parrafo.is_italic():
10                            elemento.apply_italic(parrafo)
11                            continue # GlacialNote: Appears to be unnecessary.
12                        else:
13                            parrafo.normalize_format()
14                    else:
15                        # GlacialNote: Good spot to flatten or extract to functions
16                        for img in elemento.media:
17                            if img.is_fit(slide):
18                                if media_utils.needs_light_adjustment(img):
19                                    slide.add_img(img, bg=(255,255,255))
20                                else:
21                                    raise NotImplementedError('फिट नहीं होता')
22                            elif img.need_resize():
23                                img.resize_for(slide)
```

## HEAVY DUTY EXCAVATION: FLIPPING IF-ELSE

Flipping if-else blocks is a powerful way to evaluate and flatten code.

```
1 for slide in presentation:
2     if slide.is_have_content():
3         for elemento in slide.elements:
4             if elemento.is_textual():
5                 for parrafo in elemento.parrafos:
6                     # GlacialNote: if-else main code path second
7                     if parrafo.is_have_special_format():
8                         if parrafo.is_bold():
9                             elemento.apply_bold(parrafo)
10                        elif parrafo.is_italic():
11                            elemento.apply_italic(parrafo)
12                            continue # GlacialNote: Appears to be unnecessary.
13                    else:
14                        parrafo.normalize_format()
```

## HEAVY DUTY EXCAVATION: FLIPPING IF-ELSE

Here we negate the `if` and swap the code before and after `else`.  
The normal main code path is now first.

```
1 for slide in presentation:
2     if slide.is_have_content():
3         for elemento in slide.elements:
4             if elemento.is_textual():
5                 for parrafo in elemento.parrafos:
6                     if not parrafo.is_have_special_format():
7                         parrafo.normalize_format()
8                     else:
9                         if parrafo.is_bold():
10                            elemento.apply_bold(parrafo)
11                            elif parrafo.is_italic():
12                                elemento.apply_italic(parrafo)
13                            continue # GlacialNote: Appears to be unnecessary.
```

## FLIPPING OFF THE ELSE

Now we can use `continue` to tell the else to take a hike and peel off a layer.

```
1 for slide in presentation:
2     if slide.is_have_content():
3         for elemento in slide.elements:
4             if elemento.is_textual():
5                 for parrafo in elemento.parrafos:
6                     if not parrafo.is_have_special_format():
7                         parrafo.normalize_format()
8                         continue
9                 if parrafo.is_bold():
10                    elemento.apply_bold(parrafo)
11                elif parrafo.is_italic():
12                    elemento.apply_italic(parrafo)
13                continue # GlacialNote: Appears to be unnecessary.
```

**EVOLUTIONARY  
CODESCAPING  
THE GLACIAL RETREAT**

## ENVISIONING THE FUTURE

Evolutionary Codescaping is envisioned as an integral part of software development life-cycles.

- Using GlacialNotes for continuous improvement
- Integrating refactoring seamlessly into SDLC
- Maintaining a focus on core Glacial Refactoring principles
- Tooling to facilitate understanding code behaviour and usage
- Developing parallel code paths for risk mitigation

# THE ROAD AHEAD

The journey of Glacial Refactoring is just beginning.



# ASSISTANCE WELCOME

Join me in shaping this methodology.







- Refining terminology, creating guidelines, and writing examples
- Aligning with existing practices and terminology
- Promoting adoption of a positive perspective on all kinds of past code.
- Building tooling: helper libraries, reporting tools, SDLC integrations, linting tools, etc.

# CONTACTING ROSE

Thank you for joining me on this exploration of Glacial Refactoring.

Here's ways to reach out to me.



-  Email [rose@rosehooper.com](mailto:rose@rosehooper.com)
-  Web [rosehooper.com](https://rosehooper.com)
-  Mastodon [@krayola@mastodon.social](https://mastodon.social/@krayola)
-  Github [github.com/rhooper](https://github.com/rhooper)
-  LinkedIn [linkedin.com/in/rosehooper](https://linkedin.com/in/rosehooper)
-  Discord [PyOhio https://www.pyohio.org/2023/discord/](https://www.pyohio.org/2023/discord/)