How to get away with murder refactoring

@qcmaude
This is me (and Chewbacca).

I work at #slack & live in San Francisco.
This presentation contains pictures of cake and other baked goods courtesy of The Great British Bake Off.
Agenda

1. Refactoring
   • what it is
   • why do it
2. In Practice
3. Lessons Learned
Refactoring

a definition

n. the process of restructuring existing code (e.g. the factoring) without changing its external behaviour
Monique’s Millefeuille

1. Place 1/10 inch-thick puff pastry in the oven at 350 °F.

2. Whisk the eggs, sugar, vanilla and flour in a bowl. *(white all-purpose flour is best)*

3. Add boiled milk to the bowl.

4. Return mixture to saucepan for a few minutes until it has a cream-like texture.

5. When the puff pastry is cooked, cut into three even pieces. *should take 10 minutes*

6. Layer the crème pâtissière between layers of pastry and let cool. *crème pâtissière is the cream-like mixture we just made*

Boil milk in medium saucepan.

about 15 minutes on medium heat.
but, in the world of business, it’s a little more complicated than that ...
Reasons to Refactor

1. for fun
2. out of boredom
3. “happen to be passing by”
4. to make the code more legible—or extendable—
“If it ain’t broken, don’t fix it.”
Reasons to Refactor

1. shift in product requirements
2. adopting a new technology
3. improving performance
WHAT COULD GO WRONG?
Refactoring can ...

- cause serious regressions
- unearth dormant bugs
- easily grow in scope
- introduce unnecessary complexity
So, let’s not kid ourselves ...
Refactoring

the real definition

n. the process by which we take a pile of poo and turn it into a shinier pile of poo
Narrow it down:

What’s the actual problem?
How to Get Away with Refactoring

Acme Sites
- Noelle Kelly

Channels
- # announcements
- # cs-marketing
- # cs-sales
- # feedback
- # product
- # proj-coupons
- # team-cs

Direct Messages
- slackbot
- Noelle Kelly
- Steve Young
- Terrance Perez

#proj-coupons

Noelle Kelly
Are we still on track for the new offers?

Steve Young
We are! Here's the schedule:

Coupons launch plan
314kB PDF

Noelle Kelly
Awesome. Happy to help with the rollout.
## By the numbers

<table>
<thead>
<tr>
<th># of public &amp; private channels on top five teams*</th>
<th># of channels on top five teams*</th>
<th># of channel memberships on top five users*</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 301</td>
<td>5 841 454</td>
<td>8 338 590</td>
</tr>
<tr>
<td>69 296</td>
<td>1 926 918</td>
<td>102 569</td>
</tr>
<tr>
<td>60 029</td>
<td>1 819 719</td>
<td>100 647</td>
</tr>
<tr>
<td>55 043</td>
<td>1 527 894</td>
<td>66 311</td>
</tr>
<tr>
<td>49 697</td>
<td>1 524 953</td>
<td>61 101</td>
</tr>
</tbody>
</table>

* only looks at totals on a single team (non-aggregate view of Enterprise customers)
We have two tables that store nearly identical information:

1. `teams_channels` stores a row for each public channel
2. `groups` stores a row for each private channel or group DM
Correspondingly, we have yet another two tables that store nearly identical information:

1. **teams_channels_members** stores a row for each user’s membership in a *public* channel

2. **groups_members** stores a row for each user’s membership in a *private* channel or *group* DM
# Slack in mid-2017

<table>
<thead>
<tr>
<th>Channel</th>
<th>Public Channels</th>
<th>Private Channels</th>
<th>DMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Membership</td>
<td>teams_channels</td>
<td>groups</td>
<td>teams_ims</td>
</tr>
<tr>
<td></td>
<td>teams_channels_members</td>
<td>groups_members</td>
<td></td>
</tr>
<tr>
<td>C123456</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G123456</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D123456</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Slack in mid-2017

How to Get Away with Refactoring

@qcmaude
We end up with a ton of similar queries to two tables and lots of

```
UNION
UNION ALL
LEFT (OUTER) JOIN
```

which isn’t great for performance 😞
SQL Performance 102

- **UNION** removes duplicate records.
- **UNION ALL** returns all columns (no extra distinctness check).
- **LEFT OUTER JOIN** is faster than **LEFT INNER JOIN**.
- Use **EXPLAIN** all day every day.
do some stuff

```sql
$sql = "SELECT m1.channel_id, 'C' AS type, status, archive_date, delete_date
FROM channels AS c
JOIN members AS m1 ON
  m1.channel_id=c.id AND
  m1.team_id=t.team_id AND
  m1.user_id=$user_id AND
  m1.delete_date=0
WHERE
  t.team_id=t.team_id AND
  m1.type=%public_chan_type%AND
  m1.join_date IN (%list:join_date)
UNION ALL
SELECT m2.channel_id, 'G' AS type, 'O' AS status, g.archive_date, g.delete_date
FROM groups AS g
JOIN members AS m2 ON
  m2.channel_id=g.id AND
  m2.team_id=t.team_id AND
  m2.user_id=$user_id AND
  m2.delete_date=0
WHERE
  g.team_id=t.team_id AND
  m2.type=%private_chan_type% AND
  m2.join_date=%private_chan_privacy%"

return db_fetch_team($team, $sql, array(
  'team_id'=> $team['id'],
  'user_id'=> $user['id'],
  'public_chan_type'=> channel_type_t::PUBLIC, channel_privacy_type_t::PRIVATE,
  'private_chan_type'=> channel_type_t::G,
  'public_chan_privacy'=> channel_privacy_type_t::PUBLIC, channel_privacy_type_t::PRIVATE,
));
```

do some other stuff
Get some context:

Why was it designed this way originally?
1. **Security**: keeping *private* channel information in a separate table isolates it

2. **Product history**: channels and private channels seemed like vastly different concepts

3. **Inability to travel into the future**
Put on your thinking cap:

**Brainstorm a Solution**

& **Identify the Challenges**
Remember this?

- Public channels: teams_channels
- Private channels: groups
- DMs: teams_im

- Channel membership:
  - teams_channels_members: C123456
  - groups_members: G123456
  - teams_im: D123456
Consolidate into a single `channels_members` table
Let’s do this!

channel

public channels
- teams_channels

private channels
- groups

DMs
- teams_ims

membership

channels_members

C123456

G123456

D123456

How to Get Away with Refactoring

@qcmaude
It might not be so simple ...

- SQL queries are scattered throughout the code
- About 400 callsites
- ✋ embedded in old, crufty code from 3 years ago
- Little to no unit test coverage
Sell, sell, sell:

Convince your team!
1. Our biggest customers won’t be able to handle the additional latency in a few months

2. Opportunity to easily modify a key table

3. Develop a pattern for future data consolidations
Map it out:

Write a Detailed Plan of Action
GROUPS_MEMBERS deprecation plan

1. Write a simple `lib_ud_channel_membership` library to handle most read and write cases. Exclude complicated queries with joins. Ensure that there are unit tests for every function given a local group, a local channel, a shared group and a shared channel as input. Here are some open questions with regards to the unidata library:
   a. Do we adopt `channel_privacy_types` for channels and replace `is_private`, `is_main` with just a single column to denote its privacy type? If we were to replace the `is_private` and `is_main` columns, we’d still output entirely identical information to the clients. This change would only affect the data access and very little of the webpage PHP code.
   b. New table called `channels_members`; this would have the same columns as `teams_channels_members` with an additional column for `channel_type` and `channel_privacy_type`.

2. Inform the data analytics and SLI teams of these changes; give them ample time to convert their pipelines to conform to the new system. Check that automated QA is testing the old endpoints (groups, join, etc); this’ll give us a better sense of whether any changes have broken our APIs. Notify the Vital team that they should not focus on attempting to migrate `teams_channels_members` nor `groups_members`.

3. Convert callsites one at a time. This effort would be advertised to the entire AppEng team for help converting callsites to read from the unidata library. Add new functions to the unidata library as needed.

4. Create a “copy” of `teams_channels_members` named `channels_members`; this new table will be used as the target for double writing instead of double writing new `groups_members` entries and updates to the `teams_channels_members` table. This table would have the exact same structure as that of `teams_channels_members` plus an addition `channel_type` column.

5. Once all callsites have been converted, begin double writing first in dev, then to TS, and then to all teams in production.

6. Write a consistency checker to ensure that all rows being double written to the `teams_channels_members` (or `channels_members`) table match the corresponding entry in
Be careful:
don’t bite off more than you can chew!
A few other tips

1. Get feedback from folks on other teams

2. Think outside your codebase: could your changes affect other services, other folks’ pipelines, third-party developers, etc.

3. Be generous in your estimates
Go, go, go:

Execute!
Feature flags are your friend.
Bugs are inevitable.
In case you’d forgotten 🙄

<table>
<thead>
<tr>
<th>Channel</th>
<th>Public Channels</th>
<th>Private Channels</th>
<th>DMs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>teams_channels</td>
<td>groups</td>
<td>teams_ims</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Membership</th>
<th>channels_members</th>
<th>C123456</th>
<th>G123456</th>
<th>D123456</th>
</tr>
</thead>
</table>

How to Get Away with Refactoring

@qcmaude
### One step further

<table>
<thead>
<tr>
<th>Channel</th>
<th>Public Channels</th>
<th>Private Channels</th>
<th>DMs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>channels</td>
<td>channels_members</td>
<td>teams_ims</td>
</tr>
<tr>
<td>Membership</td>
<td>C123456</td>
<td>G123456</td>
<td>D123456</td>
</tr>
</tbody>
</table>

**How to Get Away with Refactoring**

@qcmaude
1. **Dark mode:** Read from both tables, compare the results, return the value from the *old table*.

2. **Light mode:** Read from both tables, compare the results, return the value from the *new table*. 
1. Dark mode in dev environments.
2. Dark mode in production for a few weeks.
3. Light mode in dev for a two weeks.
4. Light mode to our team for one week.
5. Light mode to increasing % of teams in production over a 2 week period.
~8.2 million unique DMs

“I’m afraid I don’t understand, I’m sorry!”
Soft Deletion

team_id channel_id user_id ... date Joined ... date_deleted
Do the boogie:

Celebrate!
You’re not done yet: Clean Up
In Review

Refactoring can be a win-win for both engineers & your business but ... it has to be carefully scoped & planned out.
Piece of cake.
Merci!

Send questions, puns, & concerns to @qcmaude

And that's really the end of my attention span.