hidden mysteries behind big mobile codebases.

'reloaded edition'

@fernando_cejas

Meet @fernando_cejas

→ *Curious learner*

→ *Software engineer*

→ *Speaker*

→ *Works at @soundcloud*

→ *fernandocejas.com*

This begins with a story...

→ You are a happy developer

→ You have a lightweight pet project

→ You are the only maintainer

One man Development Process Model.

At some point in time...

→ Project starts to grow...

→ More features are required...

→ You are extremely happy for its success...

First problem: Success!

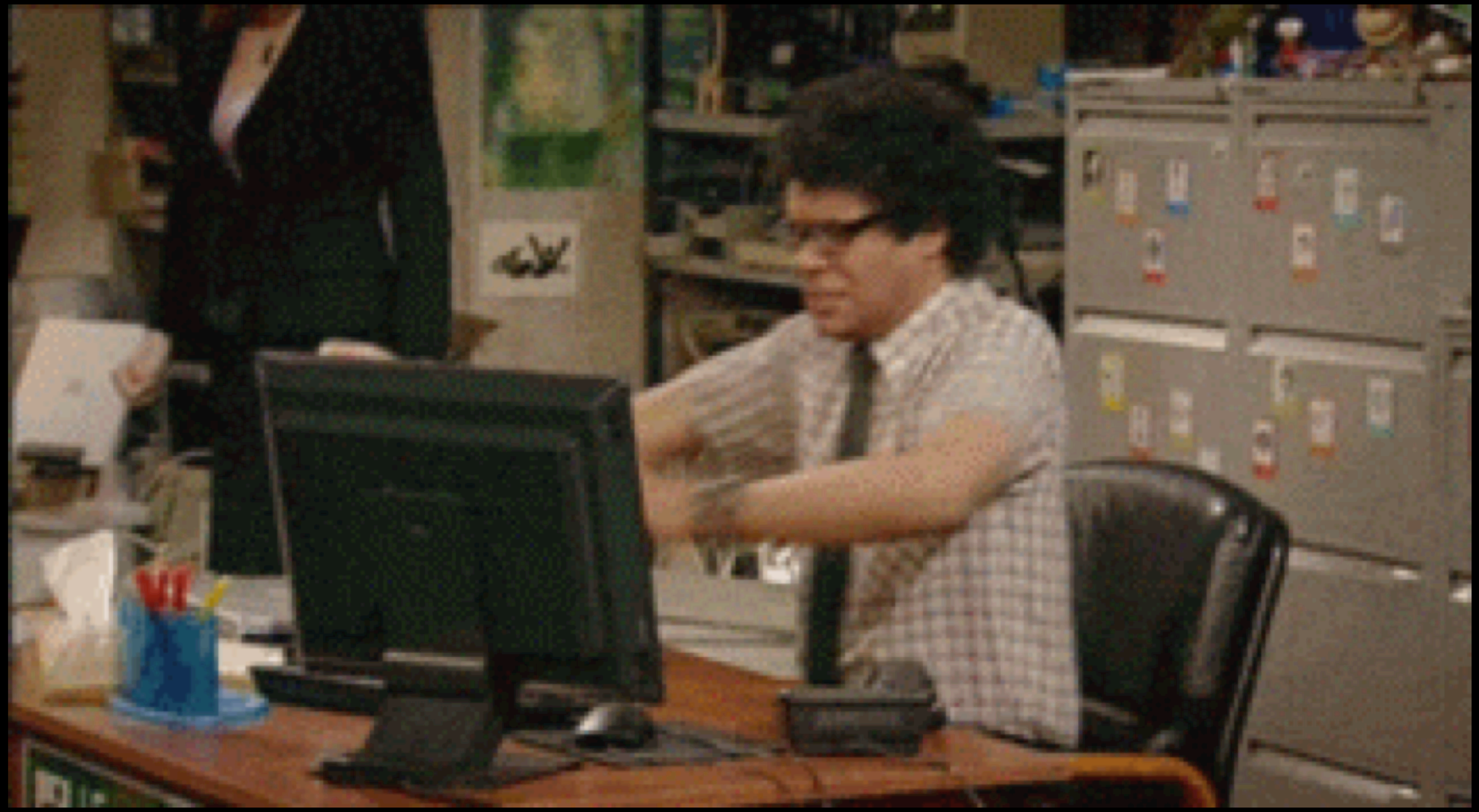
```
Android total installs: 161.854.238
iOS total installs: 122.423.077
```

Success under the hoods

→ Codebase grows.

→ No tests.

→ Inconsistency across the code.



Unsustainable situation. Why?

→ More requirements/features.

→ More contributors.

→ Time to market and dealines.

→ Complexity going up.

Many questions to answer.

→ Can we add a new functionality fast?

→ Is our codebase prepare to scale?

→ Is it hard to maintain?

→ What about technical debt?

→ How to keep it healthy and sane?

→ Is it easy to onboard new people?

→ What about our team organization?

Fact #1

If your codebase is hard to work with...then change it!

Soundcloud

→ From a monolith to a microservices architecture.

→ How we evolved our codebase on the server side.

Soundcloud App Repo DEMO.

What can we do in terms of...

→ Codebase.

→ Team Organization.

→ Working culture.

→ Processes.

...to support big mobile code bases?[1]

[1] Disclaimer: no silver bullets.

Our Codebase and its worst enemies...

Size

```
Methods in app-dev-debug.apk: 95586
Fields in app-dev-debug.apk: 61738
Lines of code: 137387
```

Complexity

```
private Node delete(Node h, Key key) {
    // assert get(h, key) != null;

    if (key.compareTo(h.key) < 0)  {
        if (!isRed(h.left) && !isRed(h.left.left))
            h = moveRedLeft(h);
        h.left = delete(h.left, key);
    }
    else {
        if (isRed(h.left))
            h = rotateRight(h);
        if (key.compareTo(h.key) == 0 && (h.right == null))
            return null;
        if (!isRed(h.right) && !isRed(h.right.left))
            h = moveRedRight(h);
        if (key.compareTo(h.key) == 0) {
            Node x = min(h.right);
            h.key = x.key;
            h.val = x.val;
            // h.val = get(h.right, min(h.right).key);
            // h.key = min(h.right).key;
            h.right = deleteMin(h.right);
        }
        else h.right = delete(h.right, key);
    }
    return balance(h);
}
```

Flaky tests

ClassName	TestName	FailureCount
com.soundcloud.android.tests.stations.StationHomePageTest	testOpenStationShouldResume	7
com.soundcloud.android.tests.stream.CardEngagementTest	testStreamItemActions	4
com.soundcloud.android.tests.stations.RecommendedStationsTest	testOpenSuggestedStationFromDiscovery	3
com.soundcloud.android.tests.player.ads.VideoAdsTest	testQuartileEvents	2
com.soundcloud.android.tests.player.ads.VideoAdsTest	testTappingVideoTwiceResumesPlayingAd	2
com.soundcloud.android.tests.player.ads.AudioAdTest	testQuartileEvents	2

Anti-patterns

```
public class NotificationImageDownloader extends AsyncTask<String, Void, Bitmap> {
    private static final int READ_TIMEOUT = 10 * 1000;
    private static final int CONNECT_TIMEOUT = 10 * 1000;

    @Override
    protected Bitmap doInBackground(String... params) {
        HttpURLConnection connection = null;
        try {
            connection = (HttpURLConnection) new URL(params[0]).openConnection();
            connection.setConnectTimeout(CONNECT_TIMEOUT);
            connection.setReadTimeout(READ_TIMEOUT);
            return BitmapFactory.decodeStream(connection.getInputStream());

        } catch (IOException e) {
            e.printStackTrace();
            return null;

        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }
}
```

Technical Debt

```
public class PublicApi {
    public static final String LINKED_PARTITIONING = "linked_partitioning";
    public static final String TAG = PublicApi.class.getSimpleName();

    public static final int TIMEOUT = 20 * 1000;
    public static final long KEEPALIVE_TIMEOUT = 20 * 1000;


    public static final int MAX_TOTAL_CONNECTIONS = 10;

    private static PublicApi instance;

    @Deprecated
    public PublicApi(Context context) {
        this(context,
             SoundCloudApplication.fromContext(context).getAccountOperations(),
             new ApplicationProperties(context.getResources()), new BuildHelper());

    }

    @Deprecated
    public PublicApi(Context context, AccountOperations accountOperations,
                     ApplicationProperties applicationProperties, BuildHelper buildHelper) {
        this(context, buildObjectMapper(), new OAuth(accountOperations),
             accountOperations, applicationProperties,
             UnauthorisedRequestRegistry.getInstance(context), new DeviceHelper(context, buildHelper, context.getResources()));
    }

    public synchronized static PublicApi getInstance(Context context) {
        if (instance == null) {
            instance = new PublicApi(context.getApplicationContext());
        }
        return instance;
    }
}
```

How can we battle this enemies and conquer a large mobile code base?

Fact #2

Architecture matters:

→ New requirements require a new architecture.

→ Scalability requires a new architecture.

Pick an architecture and stick to it

→ Onion Layers

→ Clean Architecture

→ Ports and adapters

→ Model View Presenter

→ Custom combination

→ Your own

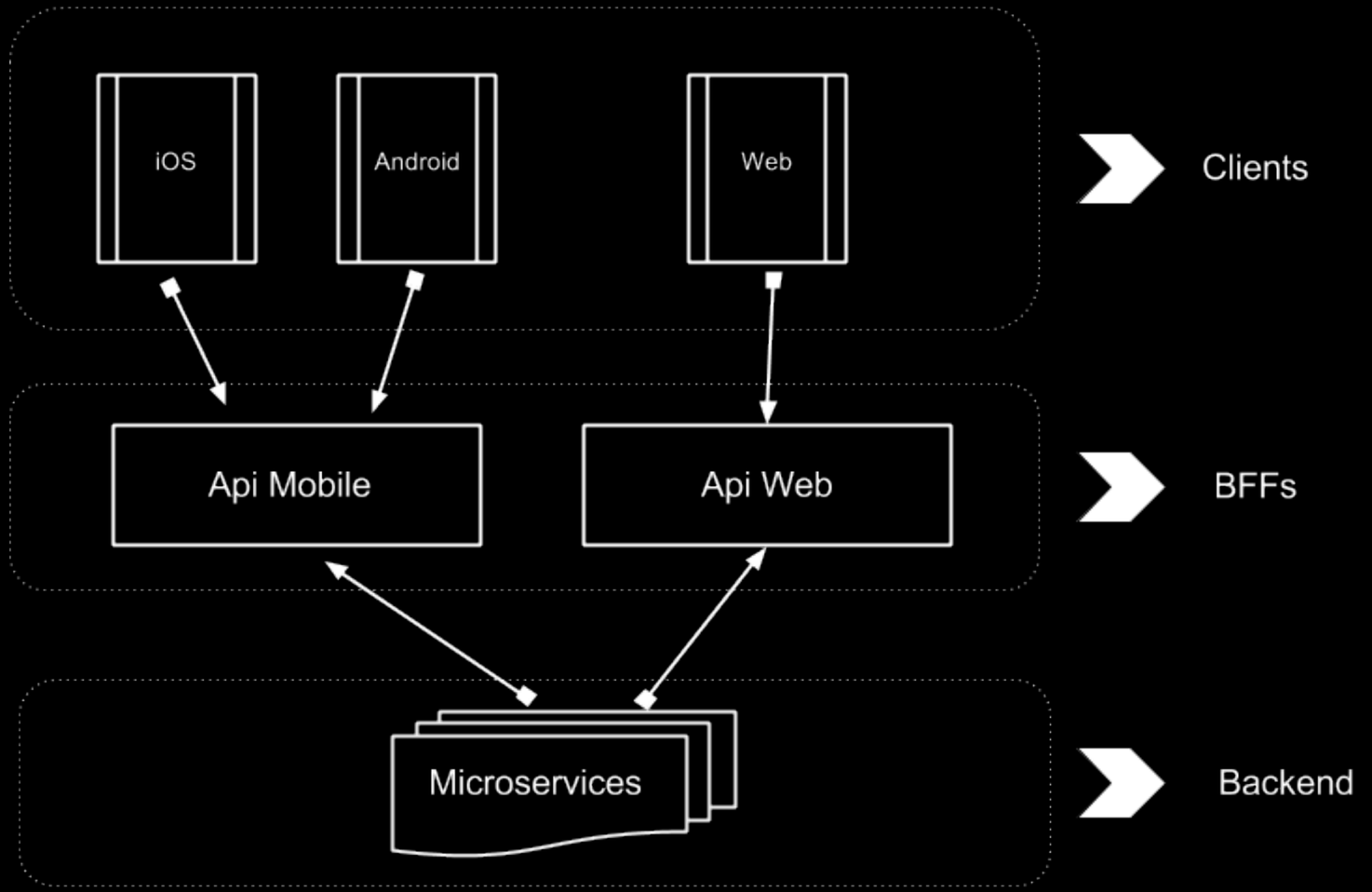
→ Sacrificial Architecture?

Benefits of a good architecture:

→ Rapid development.

→ Good Scalability.

→ Consistency across the codebase.


iOS
Android
Web
Clients
Api Mobile
Api Web
BFFs
Microservices
Backend

Architecture Android

```
public class MainActivity extends PlayerActivity {

    @Inject PlaySessionController playSessionController;
    @Inject Navigator navigator;
    @Inject FeatureFlags featureFlags;

    @Inject @LightCycle MainTabsPresenter mainPresenter;
    @Inject @LightCycle GcmManager gcmManager;
    @Inject @LightCycle FacebookInvitesController facebookInvitesController;

    public MainActivity() {
        SoundCloudApplication.getObjectGraph().inject(this);
    }

    protected void onCreate(Bundle savedInstanceState) {
        redirectToResolverIfNecessary(getIntent());
        super.onCreate(savedInstanceState);

        if (savedInstanceState == null) {
            playSessionController.reloadQueueAndShowPlayerIfEmpty();
        }
    }

    @Override
    protected void setActivityContentView() {
        mainPresenter.setBaseLayout(this);
    }

    @Override
    protected void onNewIntent(Intent intent) {
        redirectToResolverIfNecessary(intent);
        super.onNewIntent(intent);
        setIntent(intent);
    }

    private void redirectToResolverIfNecessary(Intent intent) {
        final Uri data = intent.getData();
        if (data != null
                && ResolveActivity.accept(data, getResources())
                && !NavigationIntentHelper.resolvesToNavigationItem(data)) {
            redirectFacebookDeeplinkToResolver(data);
        }
    }

    private void redirectFacebookDeeplinkToResolver(Uri data) {
        startActivity(new Intent(this, ResolveActivity.class).setAction(Intent.ACTION_VIEW).setData(data));
        finish();
    }
}
```

Architecture Android

```
public class StreamFragment extends LightCycleSupportFragment<StreamFragment>
        implements RefreshableScreen, ScrollContent {

    @Inject @LightCycle StreamPresenter presenter;

    public StreamFragment() {
        setRetainInstance(true);
        SoundCloudApplication.getObjectGraph().inject(this);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(getLayoutResource(), container, false);
    }

    @Override
    public MultiSwipeRefreshLayout getRefreshLayout() {
        return (MultiSwipeRefreshLayout) getView().findViewById(R.id.str_layout);
    }

    @Override
    public View[] getRefreshableViews() {
        return new View[]{presenter.getRecyclerView(), presenter.getEmptyView()};
    }

    @Override
    public void resetScroll() {
        presenter.scrollToTop();
    }

    private int getLayoutResource() {
        return R.layout.recyclerview_with_refresh_and_page_bg;
    }
}
```

Fact #3

Code evolution implies:

→ Constant refactoring.

→ Exploring new technologies.

→ Taking new approaches.

Refactoring

→ Code evolution.

→ Boy scouting.

→ Baby steps.

Code to refactor:

```
private void startProcessing(Map<MyKeyEnum, String> map) {
    Processor myProcessor = new Processor();
    for (Entry entry : map.entrySet()) {
        switch(entry.getKey()) {
            case KEY1:
                myProcessor.processStuffAboutKey1(entry.getValue());
                break;
            case KEY2:
                myProcessor.processStuffAboutKey2(entry.getValue());
                break;
            case KEY3:
                myProcessor.processStuffAboutKey3(entry.getValue());
                break;
            case KEY4:
                myProcessor.processStuffAboutKey4(entry.getValue());
                break;
            ...
            ...
        }
    }
}
```

Create an abstraction:

```
public interface KeyProcessor {
    void processStuff(String data);
}
```

Fill a map with implementation:

```
Map<Key, KeyProcessor> processors = new HashMap<>();
processors.add(key1, new Key1Processor());
...
...
processors.add(key4, new Key2Processor());
```

Use the map in the loop:

```
for (Entry<Key, String> entry: map.entrySet()) {
    Key key = entry.getKey();
    KeyProcessor keyProcessor = processors.get(key);
    if (keyProcessor == null) {
        throw new IllegalStateException("Unknown processor for key " + key);
    }
    final String value = entry.getValue();
    keyProcessor.processStuff(value);
}
```

Fact #4

Rely on a good test battery that backs you up.

Technical debt

"Indebted code is any code that is hard to scan."

"Technical debt is anything that increases the difficulty of reading code."

→ Anti-patterns.

→ Legacy code.

→ Abandoned code.

→ Code without tests.

Fact #5

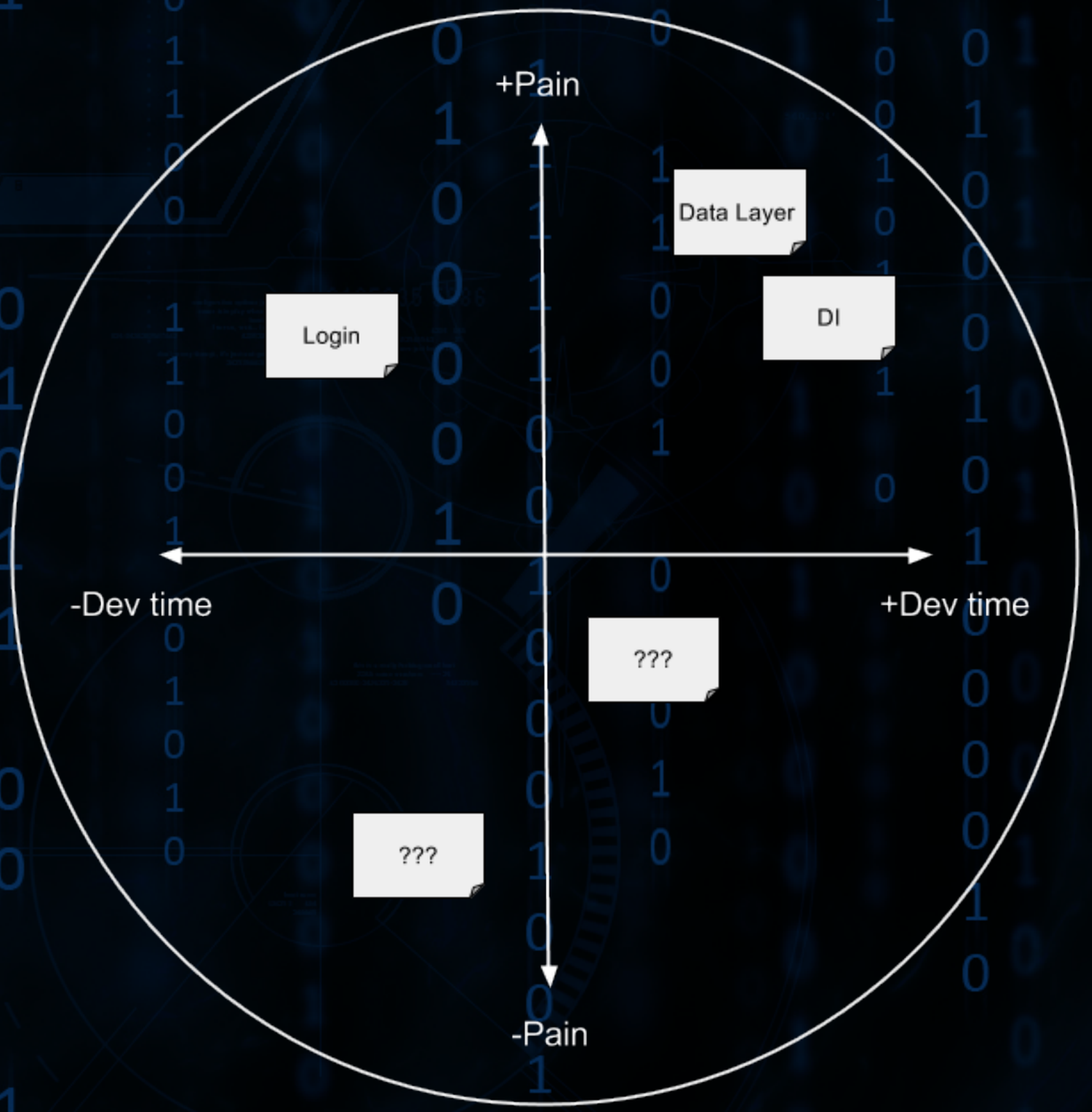
Do not let technical debt beat you.

Addressing and detecting technical debt:

→ Technical Debt Radar.

→ Static Analysis tools.

Technical Debt Radar



Fact #6

Favor code readability over performance unless the last one is critical for your business.

Perfomance

→ First rule: Always measure.

→ Encapsulate complexity.

→ Monitor it.

Fact #7

Share logic and common functionality accross applications.

At SoundCloud

→ Android-kit.

→ Skippy.

→ Lightcycle.

→ Propeller.

Fact #8

Automate all the things!

At SoundCloud

→ Continuous building.

→ Continuous integration.

→ Continuous deployment.

Lessons learned so far:

→ Wrap third party libraries.

→ Do not overthink too much and iterate.

→ Early optimization is bad.

→ Trial/error does not always work.

→ Divide and conquer.

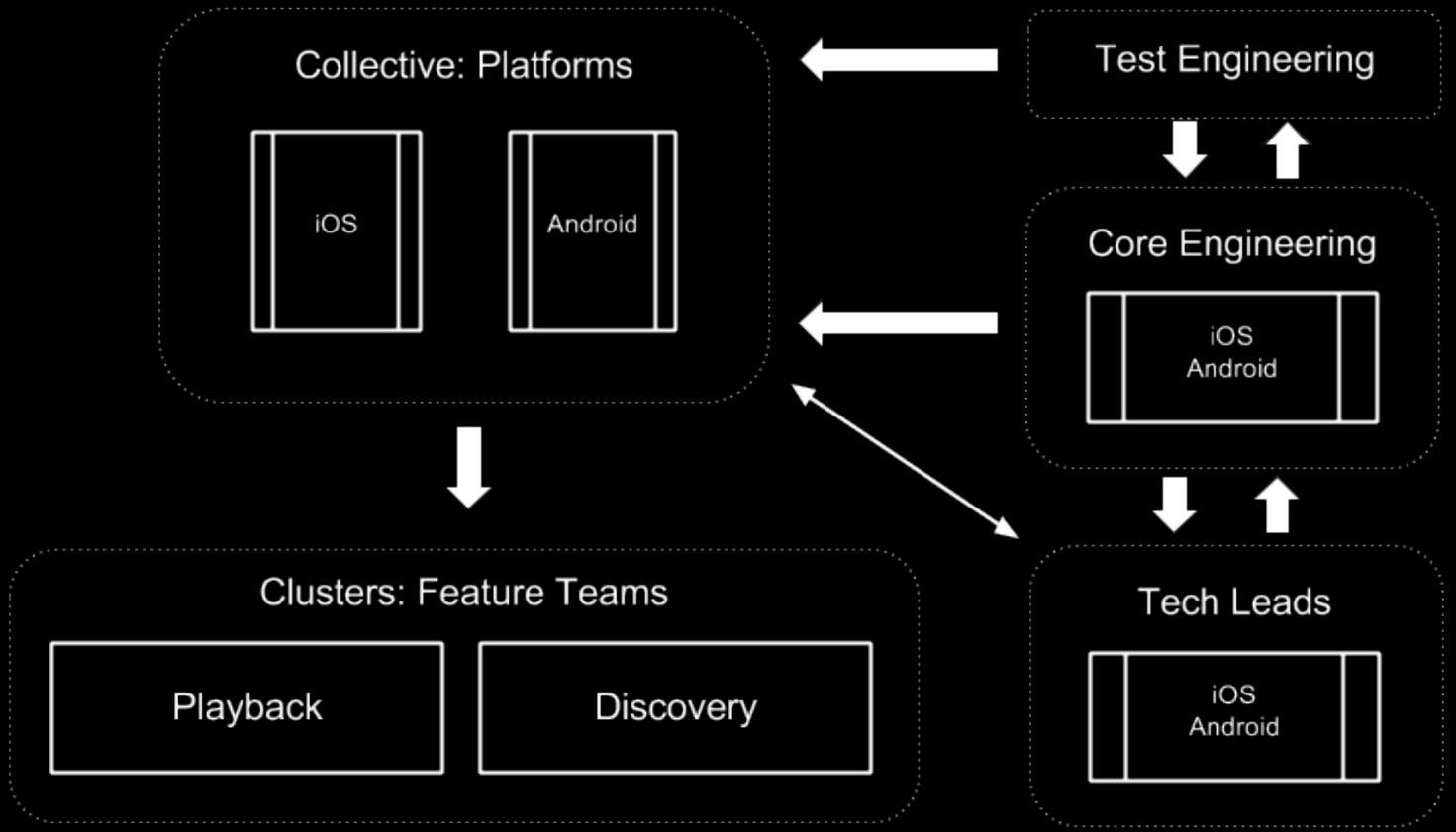
→ Prevention is better than cure.

Fact #9

Work as a team.

Team Organization

- → Platform tech lead
- → Core team
- → Feature teams
- → Testing engineering team


Collective: Platforms
iOS
Android
Test Engineering
Core Engineering
iOS
Android
Clusters: Feature Teams
Playback
Discovery
Tech Leads
iOS
Android

Working culture

→ Pair programming.

→ Git branching model.

→ Share knowledge with other platforms.

→ Agile and flexible.

→ Collective Sync meeting.

Processes

→ Onboarding new people.

→ Hiring people.

→ Sheriff.

→ Releasing: Release train model + release captains.

→ Alpha for internal use (Dog fooding).

→ Beta community.

Recap #1

→ **#1** If your codebase is hard to work with, just change it.

→ **#2** Architecture matters.

→ **#3** Code evolution implies continuous improvement.

→ **#4** Rely on a good test battery that backs you up.

→ **#5** Do not let Technical Debt beat you.

Recap #2

→ **#6** Favor code readability over performance, unless it is critical.

→ **#7** Share logic and common functionality accross applications.

→ **#8** Automate all the things.

→ **#9** Work as a team.

Conclusion

→ Use **S.O.L.I.D**

→ Software development is a **joyful ride.**

→ Make it **fun.**


Q & A

Thanks!!!

→ *@fernando_cejas*

→ *fernandocejas.com*

→ *soundcloud.com/jobs*