Open edX Conference 2023

An Opinionated Vision for Open edX Extensibility and Customization

David Joy
Learning Platform Architect at edX/2U

Goals of this talk

(Also our agenda.)

How do we extend the Open edX platform in a sustainable way?

Let’s try to answer this question.

How?

📊

1. **Define** evaluation criteria

📋

2. **Audit** our capabilities

💥

3. **Identify** problem areas

⭐

4. **Suggest** actions

Some caveats

- This is a huge topic.
- It’s complex.
- Breaking it down is difficult.
- I certainly didn’t get everything right!

This should be the beginning of a conversation.

(Trying to see the forest for the trees)

Where I'm coming from

- edX, 2019-2022: Working on the micro-frontend platform
  - MFE configurability
  - Internationalization
  - Branding and theming
  - Component customization
  - Plugins and LTI
- 2U, 2022-Present: Focused on 2U's architectural relationship to the platform

👍 *This guy* 👍 at Open edX 2019
talking about
frontend re-platforming
(that blazer tho)

There was a survey for this talk

- Sent out March 16
- Focused on **difficulty** and **rarity**
- 17 respondents (I'll take it!)
- Probably not statistically significant
- But definitely showed some patterns

📊 Define evaluation criteria

“Make the **common stuff easy** and make the **uncommon stuff possible.**”

-Steven Burch
Open edX Conference 2019 during the Theming Advisory Group meeting

Turn it into some quadrants

	Common	Rare
Hard	🎯 Invest	😐 Acceptable?
Easy	✅ Good	✋ / 📢 Stop or promote

Categories: What are we extending?

- ⚙️ Backend
- 🖼️ Frontend
- 🖋️ Content
- 💞 Cross-service

Methods: How are we extending it?

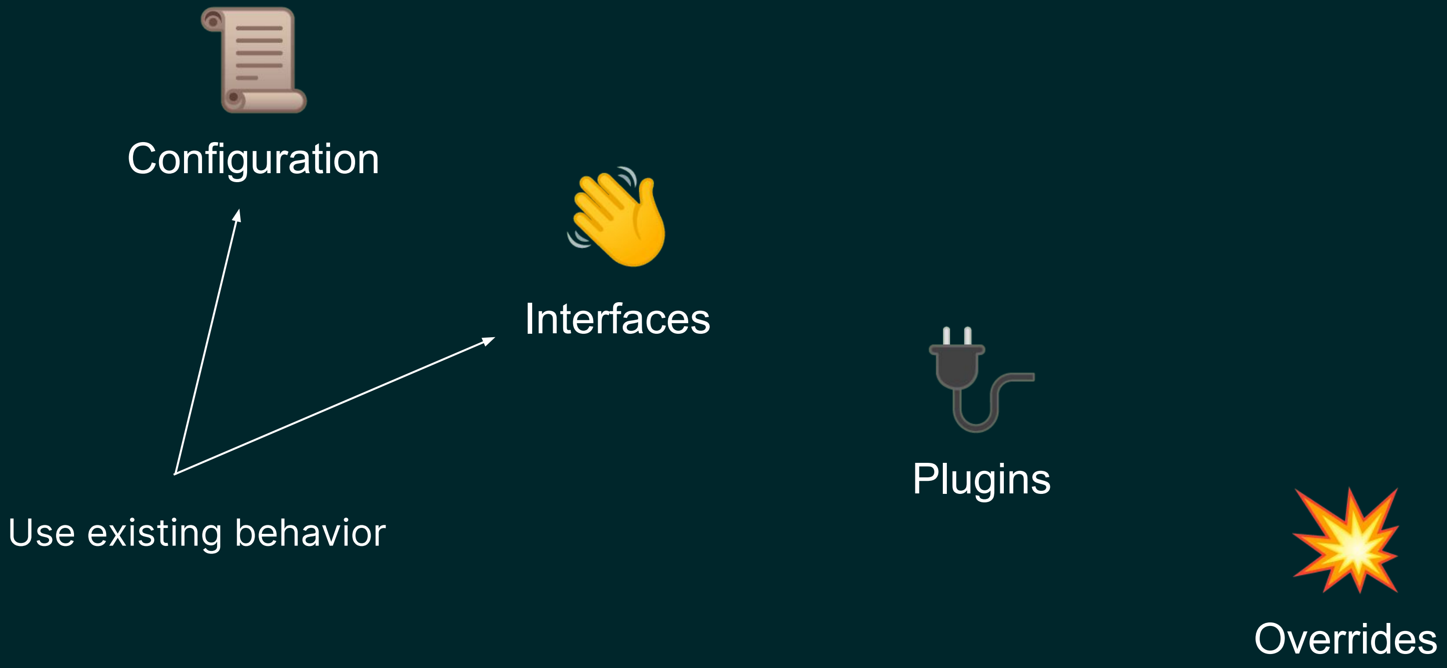
📜 Configuration

👋 Interfaces

🔌 Plugins

💥 Overrides

Methods: How are we extending it?



📜 Configuration

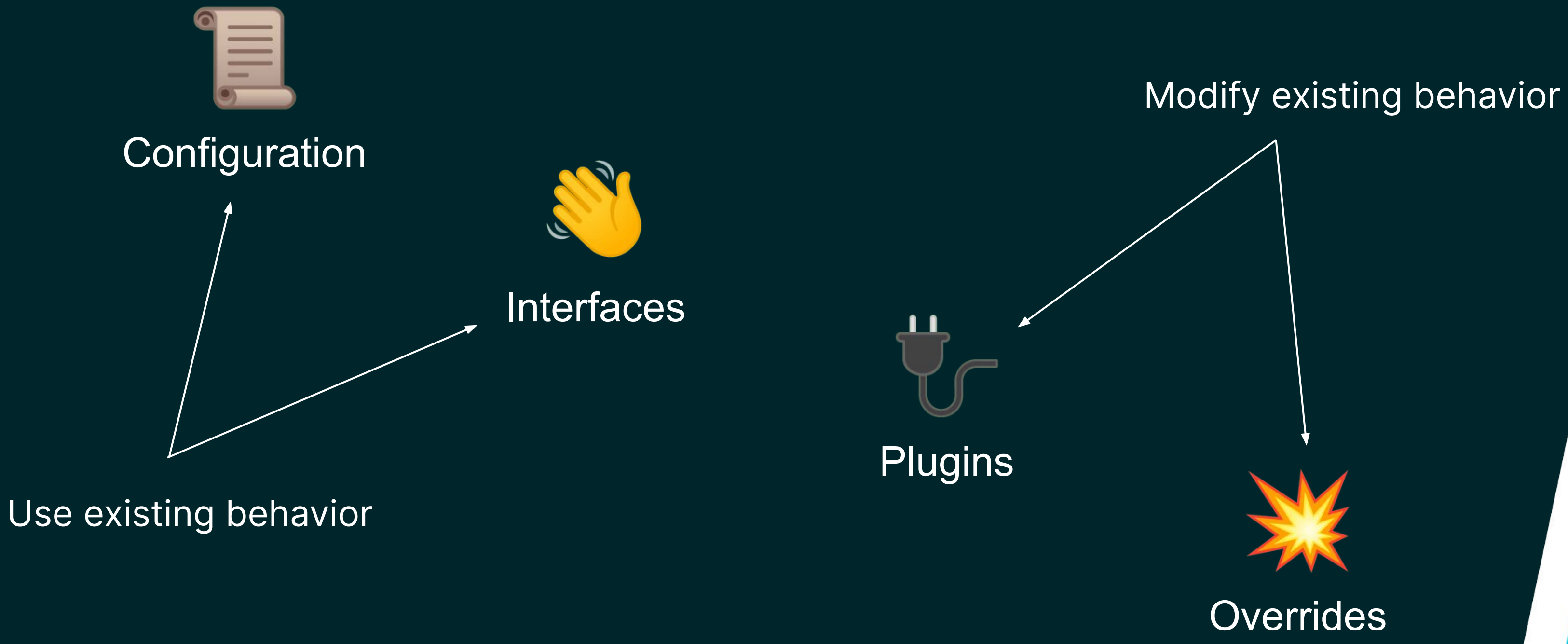
👋 Interfaces

🔌 Plugins

💥 Overrides

Use existing behavior

Methods: How are we extending it?



Configuration

Interfaces

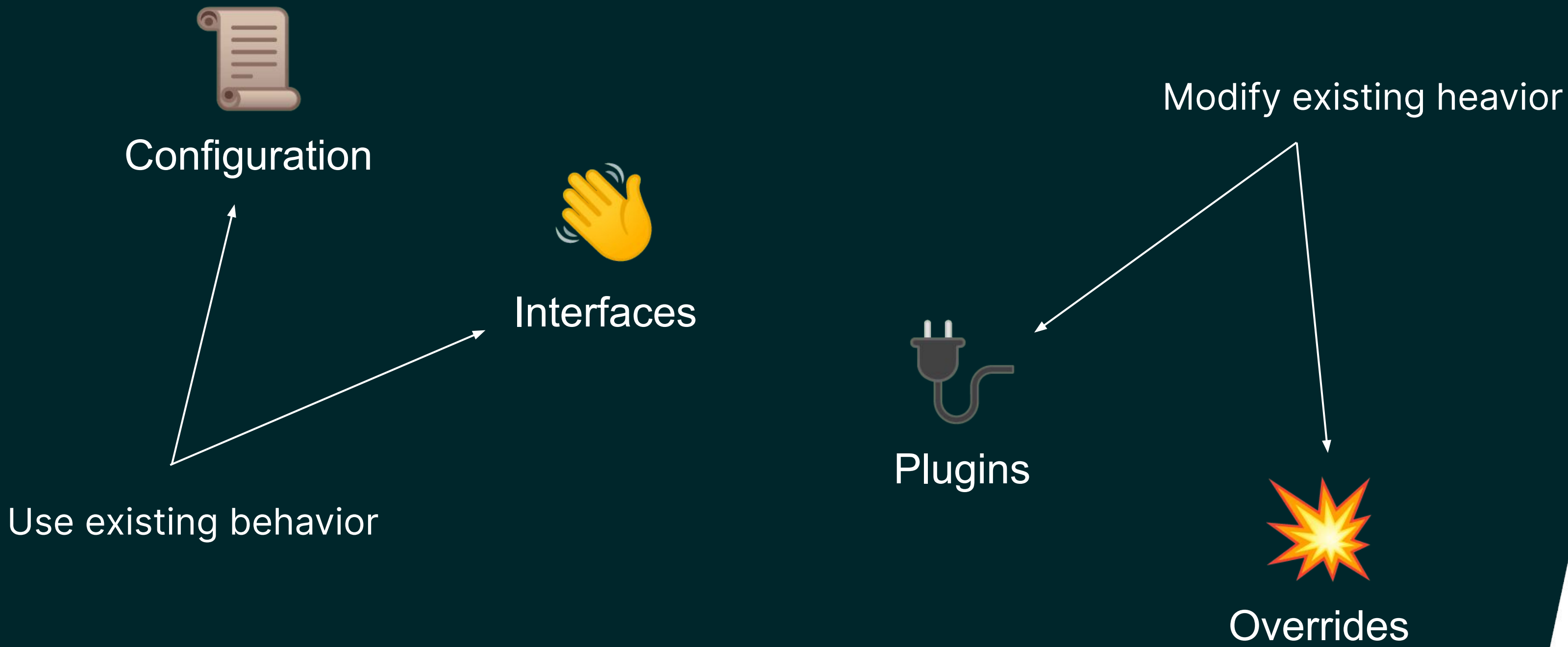
Use existing behavior

Modify existing behavior

Plugins

Overrides

Methods: How are we extending it?



Configuration

Interfaces

Use existing behavior

Modify existing heavior

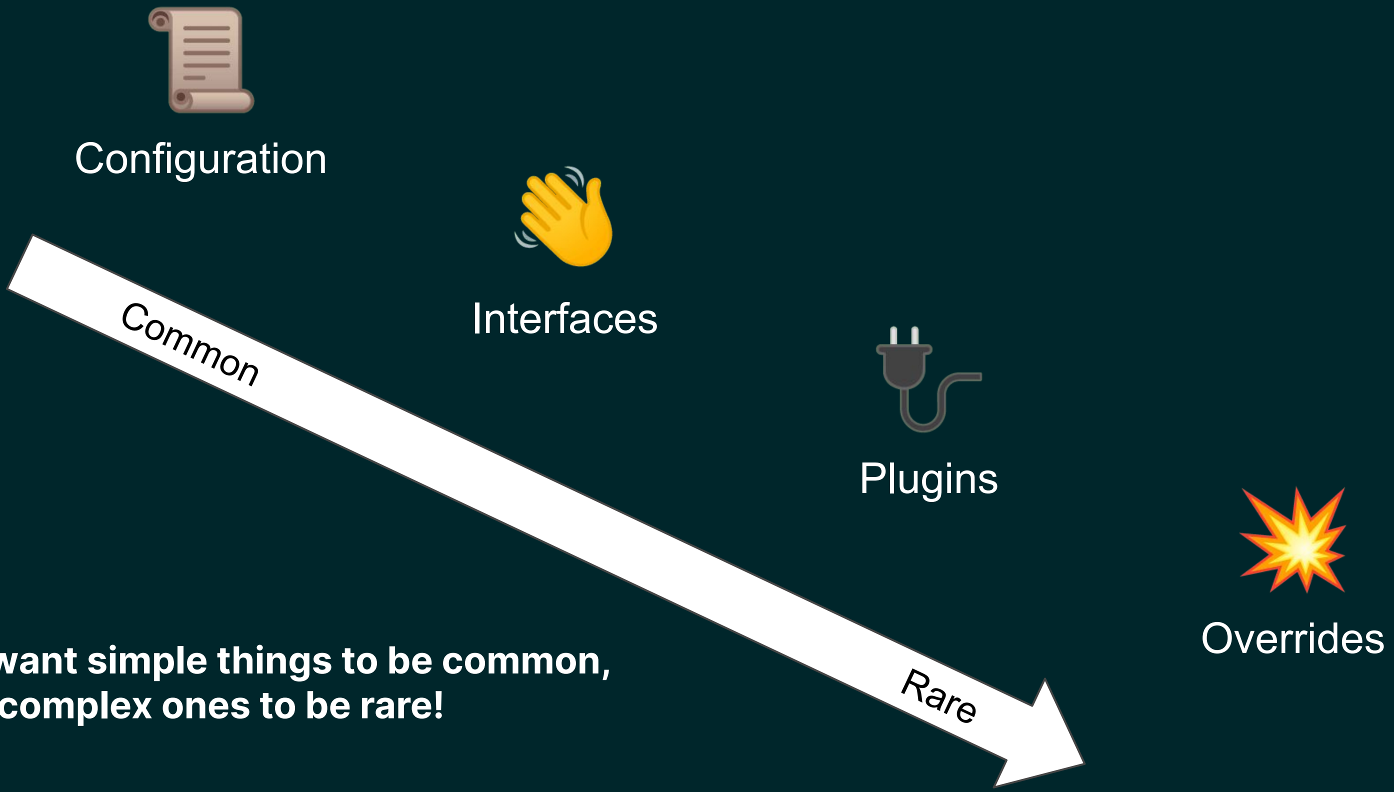
Plugins

Overrides

Deficiencies in config and interfaces result in more plugins and overrides!

Malcolm called it

Methods: How are we extending it?



📜 Configuration

👋 Interfaces

🔌 Plugins

💥 Overrides

Common

Rare

We want simple things to be common, and complex ones to be rare!

How can we evaluate ease of use?

😄 Approachability

✅ Maintainability

📚 Documentability

🛠️ Supportability

How can we evaluate ease of use?

Plugin Authors

😄 Approachability

✅ Maintainability

📚 Documentability

🛠️ Supportability

How can we evaluate ease of use?

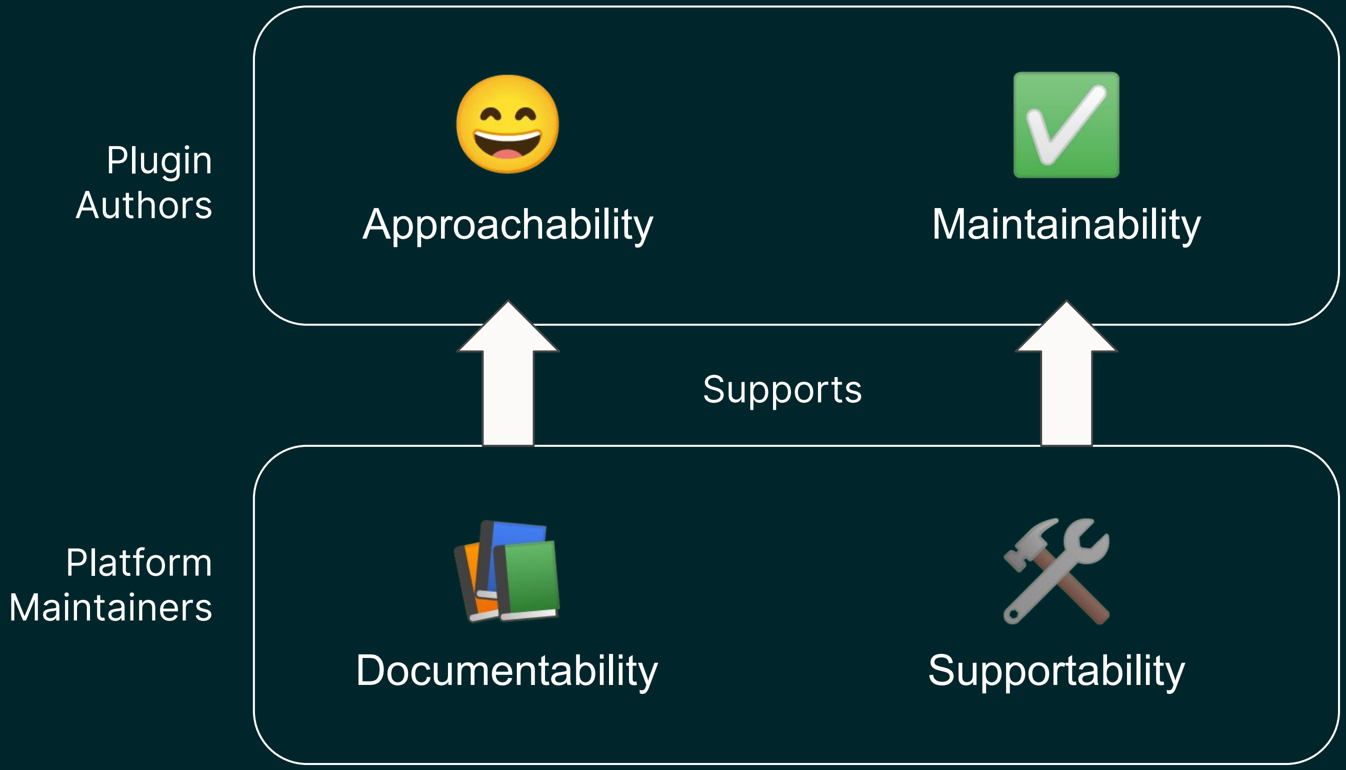
Plugin Authors

- 😄 Approachability
- ✅ Maintainability

Platform Maintainers

- 📚 Documentability
- 🛠️ Supportability

How can we evaluate ease of use?



Plugin Authors

😄 Approachability

✅ Maintainability

Supports

Platform Maintainers

📚 Documentability

🛠️ Supportability

Evaluation criteria summary

Categories	⚙️ Backend	🖼️ Frontend
	🖋️ Content	💞Cross-service
Methods	📜 Configuration	👋 Interfaces
	🔌 Plugins	💥 Overrides
Ease of Use	😄 Approachability	📚 Documentability
	🛠️ Supportability	✅ Maintainability

📋 Audit our capabilities

Audit by category

📜 1. Configuration

👋 2. Interfaces

🔌 3. Plugins

💥 4. Overrides

📜 Configuration capabilities

📜 Configuration (1 of 2)

Django Settings

⚙️ Backend

Common ▮▯▯▯ Rare   Easy 🟥▯▯ Hard

- Create a file that contains customized versions of the settings you want to configure.

- Multi-service configuration overlap and layering creates complexity.

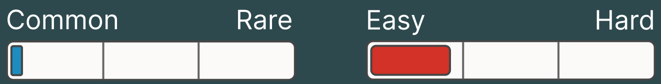
🆗 Approachability
✅ Maintainability
🆗 Documentability
❌ Supportability

How do I read this?

📜 Configuration (1 of 2)

Django Settings

⚙️ Backend



Survey results
0 = Very Easy/Common
3 = Very Hard/Rare

- Create a file that contains customized versions of the settings you want to configure.

How does it work?

- Multi-service configuration overlap and layering creates complexity.

Critique

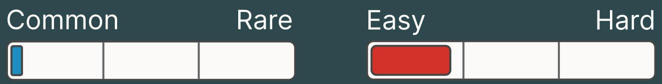
🆗 Approachability
✅ Maintainability
🆗 Documentability
❌ Supportability

Grading

📜 Configuration (1 of 2)

Django Settings

⚙️ Backend



- Create a file that contains customized versions of the settings you want to configure.

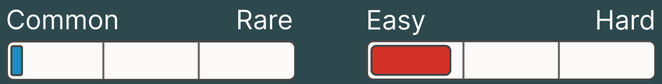
- Multi-service configuration overlap and layering creates complexity.

🆗 Approachability
✅ Maintainability
🆗 Documentability
❌ Supportability

📜 Configuration (1 of 2)

Django Settings

⚙️ Backend

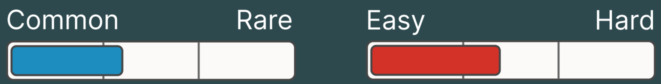


- Create a file that contains customized versions of the settings you want to configure.
- Multi-service configuration overlap and layering creates complexity.

🆗 Approachability
✅ Maintainability
🆗 Documentability
❌ Supportability

Micro-frontend Environment Variables

🖼️ Frontend



- Supply variables on the command-line when building the MFE or use the MFE config API for runtime variables.
- Only strings! Command-line is error prone and unintuitive.

❌ Approachability
🆗 Maintainability
✅ Documentability
✅ Supportability

📜 Configuration (2 of 2)

Backend Translations

⚙️ Backend

Common 🟦⬜⬜ Rare   Easy 🟥⬜⬜ Hard

- Create .po files for the translations to add, either via Transifex, Tutor, or Forking
- Tutor makes this pretty easy! Ideally services would copy, reducing complexity.

✅ Approachability
✅ Maintainability
🆗 Documentability
✅ Supportability

Micro-frontend Translations

🖼️ Frontend

Common 🟦⬜⬜ Rare   Easy 🟥🟥⬜ Hard

- Tutor can edit translations, but not add new ones. New locales not possible without forking.
- Desperately needs investment.

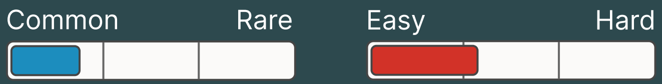
🆗 Approachability
🆗 Maintainability
❌ Documentability
❌ Supportability

👋 Interface capabilities

👋 Interface (1 of 3)

REST APIs

💞Cross-service

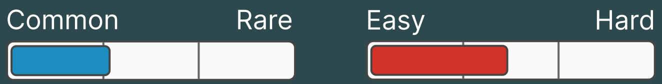


- Make a request to a known REST API endpoint.
- Doc and discoverability are current challenges. Inconsistent versioning strategy hampers maintainability.

✅ Approachability
❌ Maintainability
🆗 Documentability
🆗 Supportability

LTI

🖼️ Frontend & 🖋️ Content



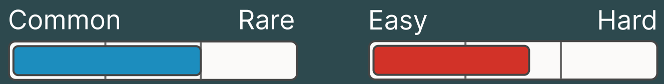
- Create a tool that satisfies the LTI spec and configure platform to launch it. Tools are sandboxed in iframe.
- It’s a standard! Issues often affect user experience. Encourage broader adoption.

❌ Approachability
✅ Maintainability
🆗 Documentability
✅ Supportability

👋 Interface (2 of 3)

Hooks Extension Framework Events

⚙️ Backend



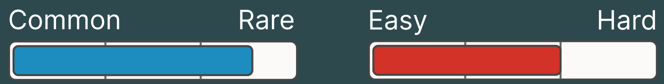
- Write a receiver in a Django App Plugin to receive Django signal-based events.

- Great work! Decouples extensions from core. Requires coding. Stay vigilant to keep events idiomatic. What about versioning?

✅ Approachability
🆗 Maintainability
✅ Documentability
✅ Supportability

Event Bus

💞Cross-service



- Write an event consumer to subscribe and process events off the bus.

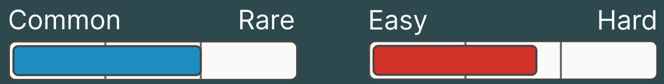
- Finally! Get on the bus! Same notes as HEFE to the left.

❌ Approachability
✅ Maintainability
🆗 Documentability
✅ Supportability

👋 Interface (3 of 3)

Custom JavaScript Problems (jsinput)

🖋 Content

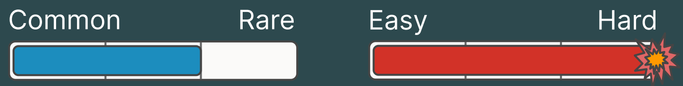


- Create JS problem, upload in Files & Uploads, configure in Studio
- Iframing these problems keeps them sandboxed and interface-like. Python in a script tag is odd.

🆗 Approachability
🆗 Maintainability
🆗 Documentability
🆗 Supportability

Micro-frontend Service Implementations

🖼 Frontend



- Write a logging or analytics implementation that satisfies the interface. *Then cry cause you can't use it.*
- The limitations in MFE env variables make it impossible to configure this short of forking.

❌ Approachability
❌ Maintainability
✅ Documentability
❌ Supportability

🔌 Plugin capabilities

🔌 Plugin (1 of 2)

XBlocks

🖋 Content

Common 🟦⬜⬜ Rare

Easy 🟥⬜⬜ Hard

- Create and install an XBlock that uses the XBlock API.
- For being so common, is a complex technique. Sandboxing is a problem.

🆗 Approachability
🆗 Maintainability
🆗 Documentability
🆗 Supportability

Django App Plugins

⚙️ Backend

Common 🟦⬜⬜ Rare

Easy 🟥⬜⬜ Hard

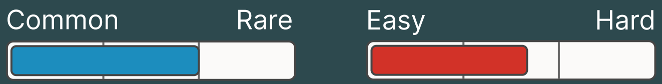
- Create a pip-installed Django app and the Django project will automatically load it.
- Reduces boilerplate and the need to fork/add custom configuration. Sandboxing is a problem.

✅ Approachability
🆗 Maintainability
✅ Documentability
✅ Supportability

🔌 Plugin (2 of 2)

Hooks Extension Framework Filters

⚙️ Backend



- Write a PipelineStep and configure to run on existing filter pipeline.

- Potentially invasive modification - powerful but risky. Discrete set of extension points.

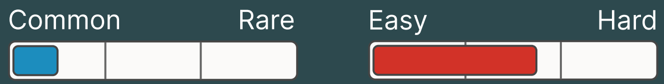
✅ Approachability
🆗 Maintainability
✅ Documentability
✅ Supportability

💥 Override capabilities

💥 Override (1 of 2)

Comprehensive Theming

⚙️ Backend & 🖼️ Frontend

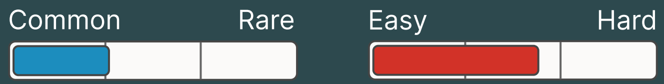


- Create and load a theme repo which is overlaid on default frontend code.
- Incredibly powerful, but invasive. Requires domain knowledge, very risky for something so important.

❌ Approachability
❌ Maintainability
🆗 Documentability
🆗 Supportability

Micro-frontend Branding

🖼️ Frontend



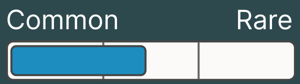
- Create brand package and alias in as brand dependency when building MFE.
- Isolated to SASS/CSS, but exposes entire stylesheet. Expose config-like subset?

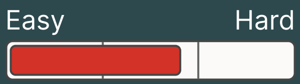
🆗 Approachability
🆗 Maintainability
✅ Documentability
✅ Supportability

💥 Override (2 of 2)

Micro-frontend Component Overrides

🖼️ Frontend





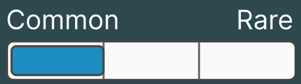
- Headers and footers. Fork package and edit, alias in as frontend-component-* dependency when building MFE.

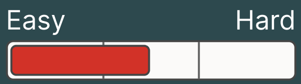
- Effectively forking but with some contracts.

❌ Approachability
🆗 Maintainability
🆗 Documentability
🆗 Supportability

Forking

🌍🌎🌏 Anything





- Fork code in git. Edit. Cry when you need to merge or rebase on upstream changes.

- Forking should be the customization mechanism of last resort - damningly common. Barrier to upgrading the platform.

❌ Approachability
❌ Maintainability
❌ Documentability
❌ Supportability

Identify problem areas

Ease of Use Scores

Scoring: ✅ 2 🆗 1 ❌ 0

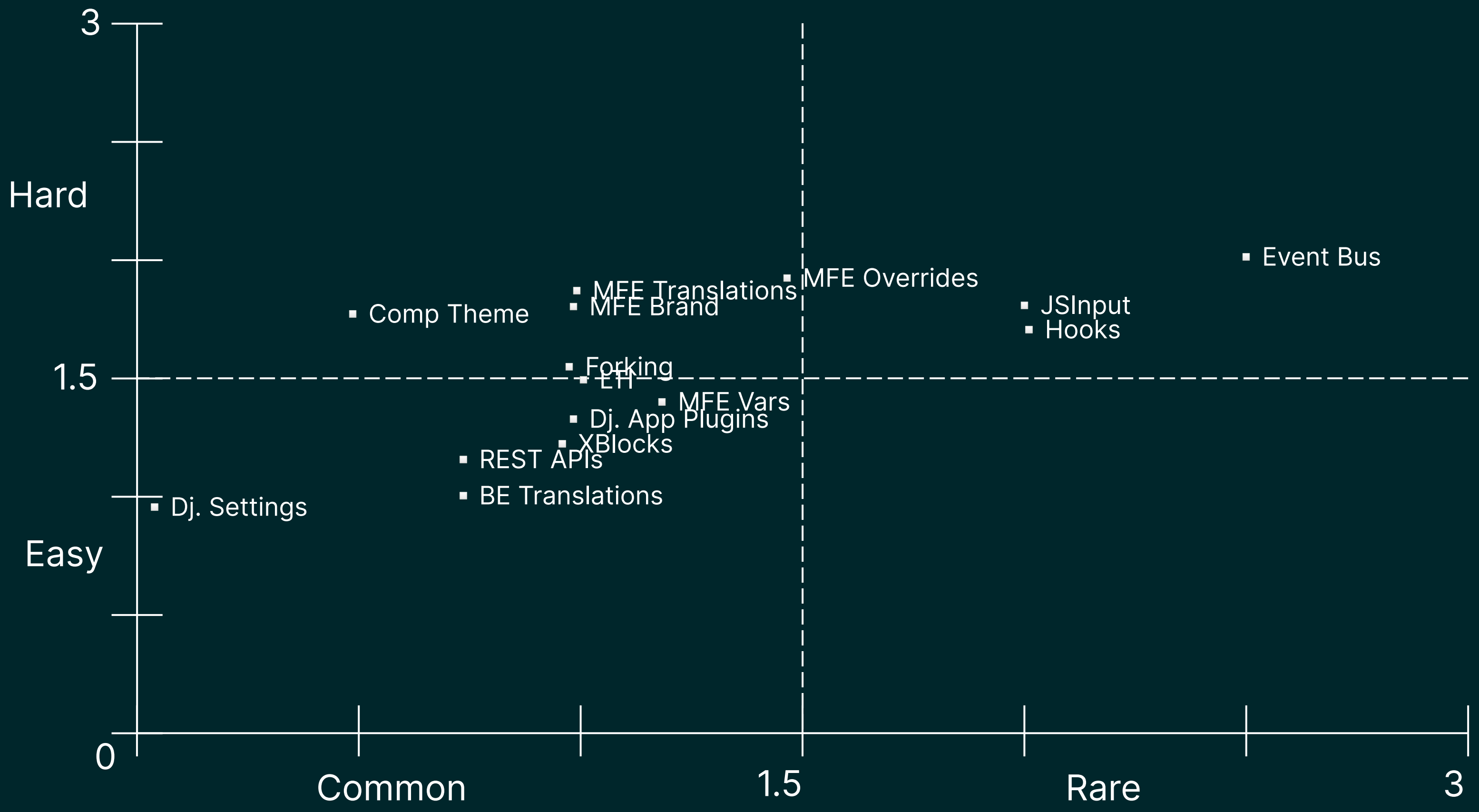
	Config	Interfaces	Plugins	Overrides
Approachability	50% (4/8)	42% (5/12)	83% (5/6)	12% (1/8)
Maintainability	75% (6/8)	50% (6/12)	50% (3/6)	25% (2/8)
Documentability	50% (4/8)	66% (8/12)	83% (5/6)	50% (4/8)
Supportability	50% (4/8)	66% (8/12)	83% (5/6)	50% (4/8)

Score Averages

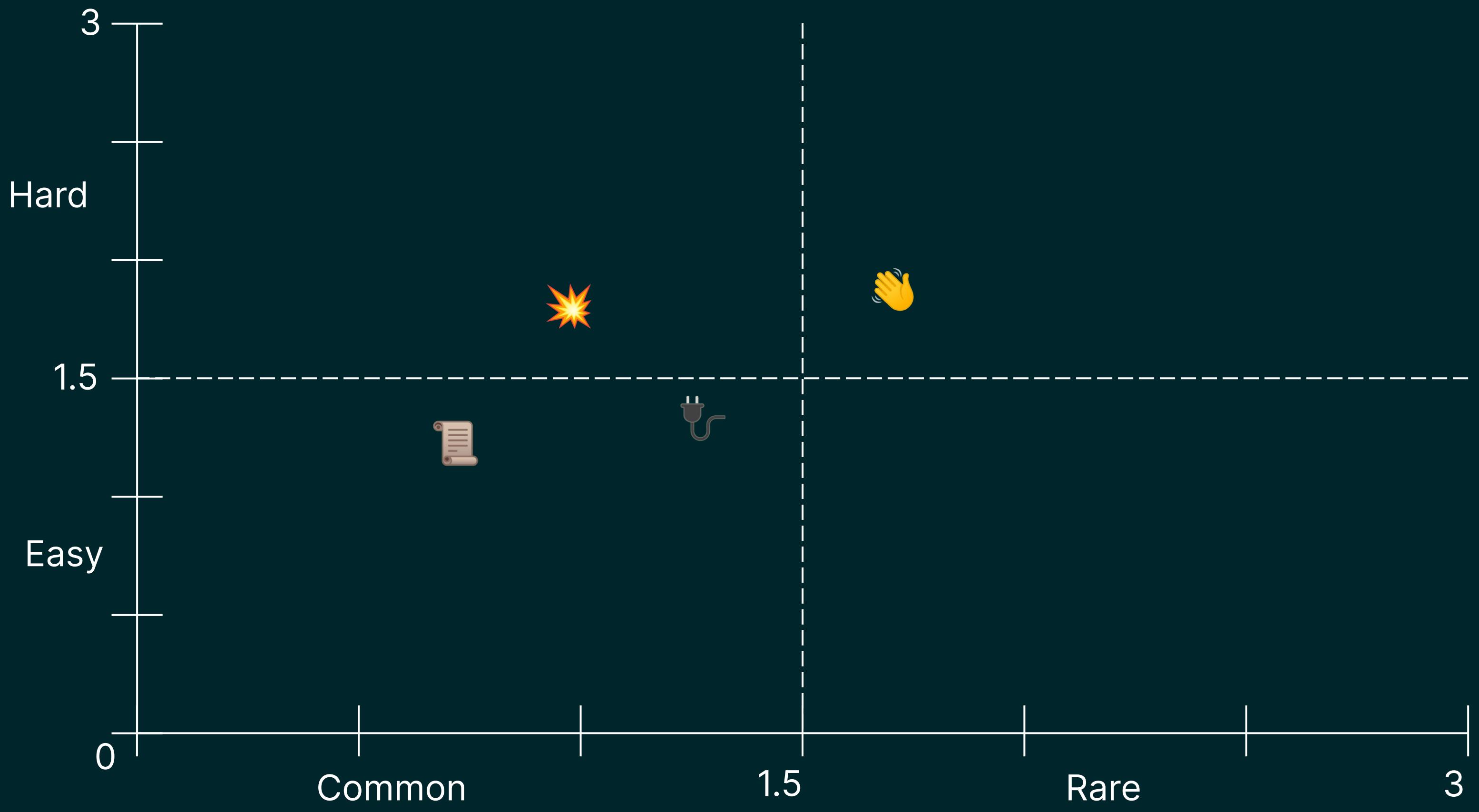
Scoring: ✅ 2 🆗 1 ❌ 0

	Config	Interfaces	Plugins	Overrides
Approachability	50% (4/8)	42% (5/12)	83% (5/6)	12% (1/8)
Maintainability	75% (6/8)	50% (6/12)	50% (3/6)	25% (2/8)
Documentability	50% (4/8)	66% (8/12)	83% (5/6)	50% (4/8)
Supportability	50% (4/8)	66% (8/12)	83% (5/6)	50% (4/8)
	56%	56%	75%	34%

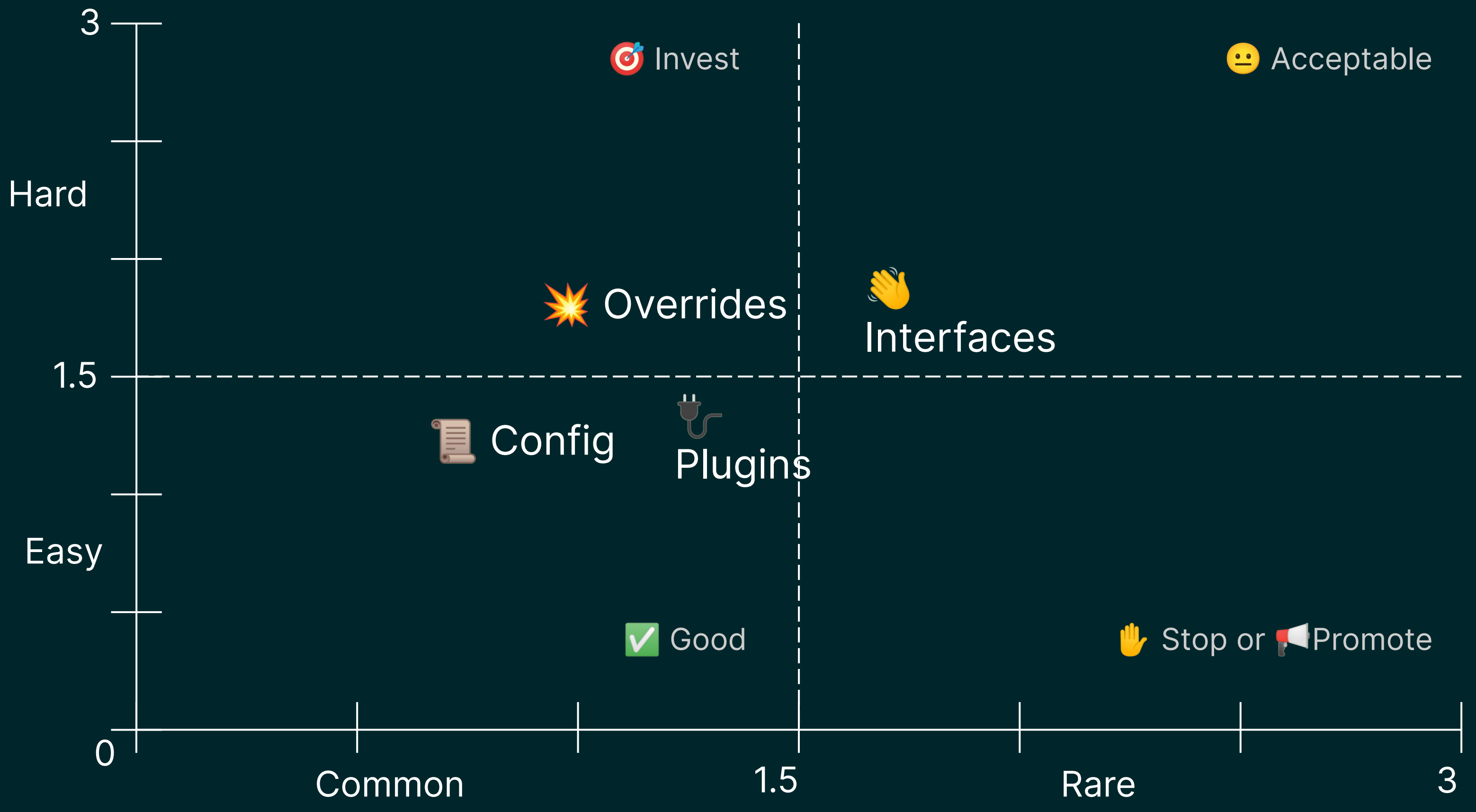
Survey Summary



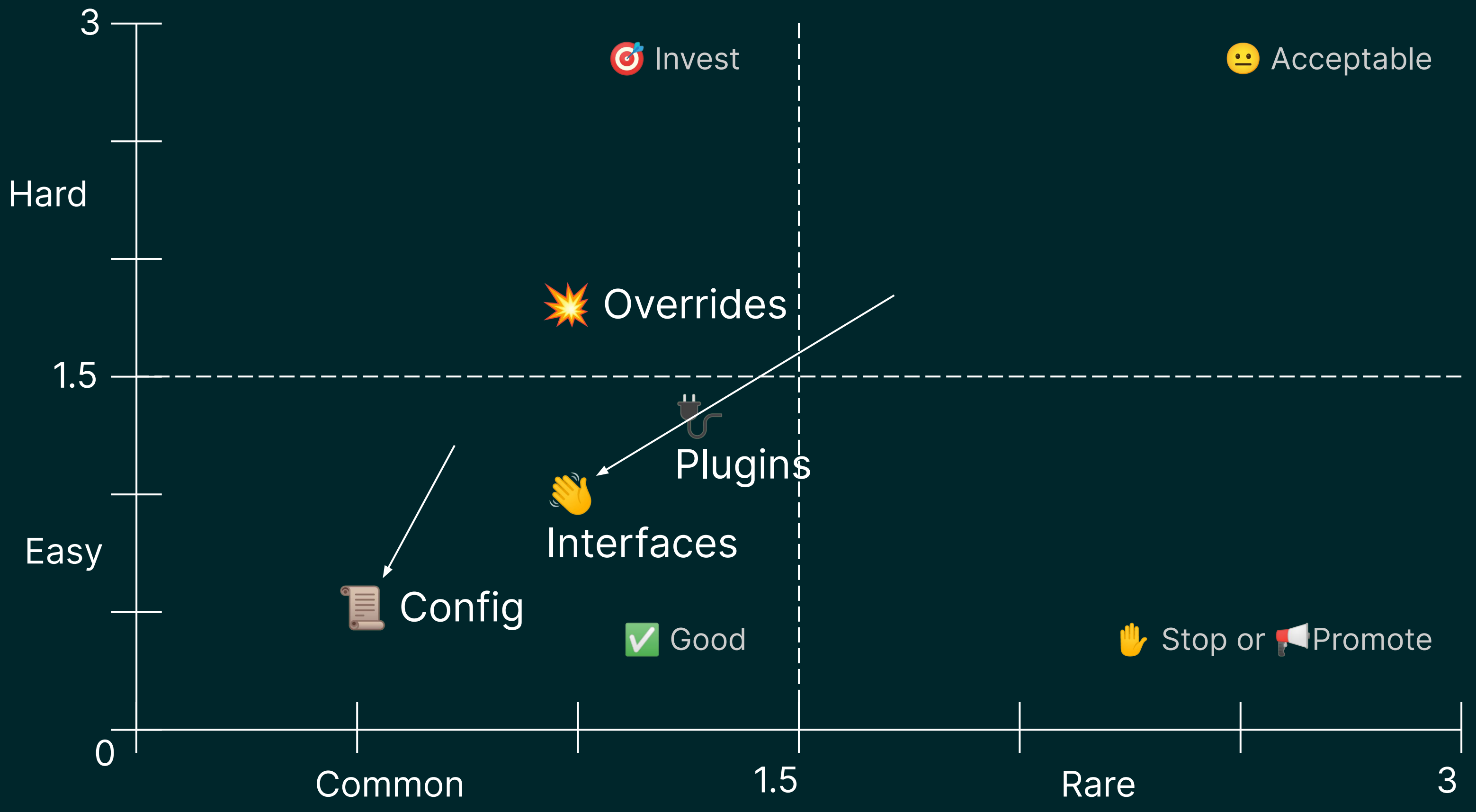
Survey Category Averages



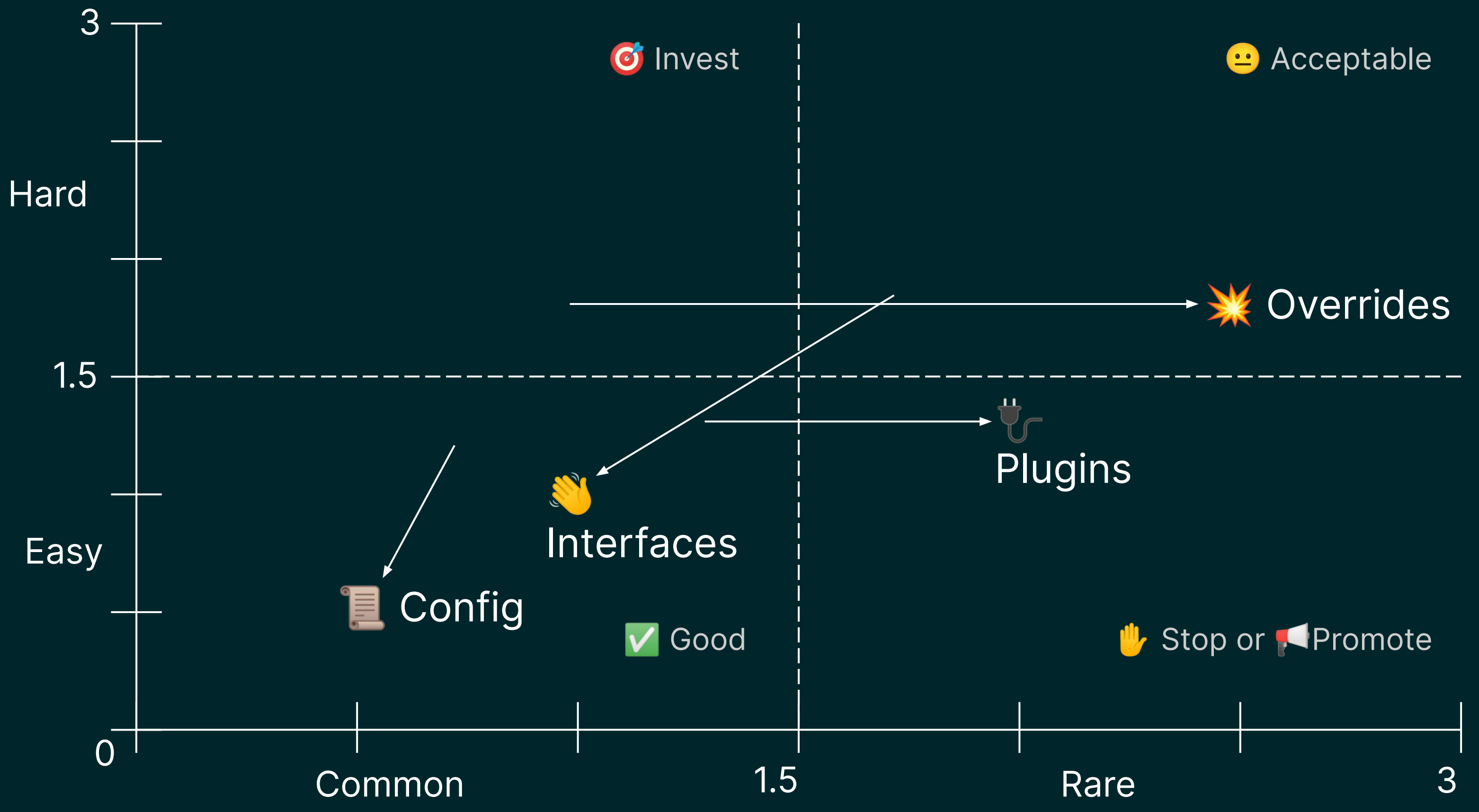
Survey Category Averages



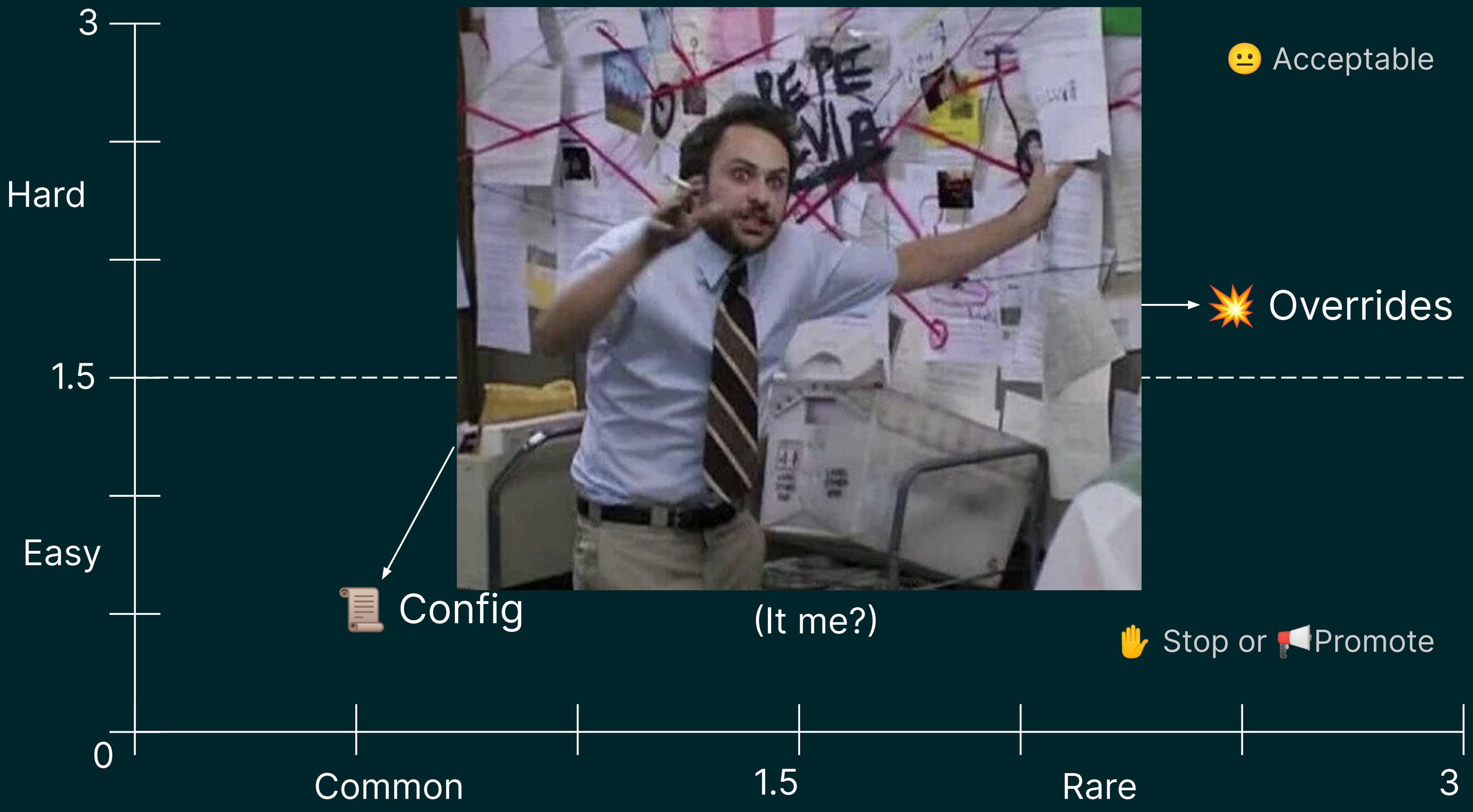
Invest in fundamentals



To make modifying less common



Oh no, am I this guy?



Coverage (Category vs. Method)

	Config	Interfaces	Plugins	Overrides
Backend	⭐⭐	⭐	⭐⭐	⭐
Frontend	⭐⭐	⭐⭐		⭐⭐
Cross-service		⭐⭐		
Content		⭐	⭐	

Coverage

	Config	Interfaces	Plugins	Overrides
Backend	⭐⭐	⭐	⭐⭐	⭐
Frontend	⭐⭐	⭐⭐		⭐⭐
Cross-service	~👍	⭐⭐		
Content	~👍	⭐	⭐	

Coverage

	Config	Interfaces	Plugins	Overrides
Backend	⭐⭐	⭐	⭐⭐	⭐
Frontend	⭐⭐	⭐⭐		⭐⭐
Cross-service	~👍	⭐⭐		🚫
Content	~👍	⭐	⭐	🚫

Coverage

	Config	Interfaces	Plugins	Overrides
Backend	⭐⭐	⭐	⭐⭐	⭐
Frontend	⭐⭐	⭐⭐	?	⭐⭐
Cross-service	~👍	⭐⭐		🚫
Content	~👍	⭐	⭐	🚫

Suggest actions

📜 Configuration

- **Django Settings**: Simplify layers
- **MFE Settings**: Allow JavaScript config
- **Backend Translations**: Service-level support for modifications
- **MFE Translations**: Dynamic list of languages, support for modifications

👋 Interfaces

- **Rest APIs**: Versioning! Discoverable documentation.
- **LTI**: Keep config simple and approachable
- **Hooks Events**: Invest in event creation best practices and standards, versioning
- **Event Bus**: Invest in cookie cutters, event creation best practices and standards, versioning
- **Custom JS Problems**: Rethink grading and modernize
- **MFE Service Implementations**: Finish the feature!

🔌 Plugins

- **XBlocks**: Modernization and simplification, frontend XBlocks, import linting
- **Django App Plugins**: Import linting to improve maintainability
- **Hooks Filters**: Document supported pipelines well, versioning

Whither Art Thou, Plugins?

- **Frontend plugins** would alleviate some of our need for frontend overrides.
- Since we don't have them, comprehensive theming, component overrides, and forking are taking their place.
- What does a **cross-service plugin** look like? Maybe pipeline-able APIs, like hooks filters but for REST APIs and events.

💥 Overrides

- **Comprehensive Theming**: Invest in MFE capabilities, deprecate
- **MFE Branding**: Design tokens, expose config-like subset of variables
- **MFE Component Overrides**: Re-think with modular MFEs, frontend plugins
- **Forking**: Flesh out other capabilities to minimize usage

Extensibility/Customization Goals

- **Configuration** to get simpler, encouraging adoption and localization.
- **Interfaces** become more approachable and maintainable to encourage their use and take pressure off of overrides.
- More options for **plugins** in the frontend to alleviate many of our needs for overrides.
- **Overrides** to become methods of last resort.

Goal is to decompose and simplify mechanisms for common needs, leaving invasive mechanisms for uncommon/invasive changes.

Capability Creation Best Practices

- Work to simplify
- Overrides: easy to support, a nightmare to maintain
- Decompose blanket overrides into config, interfaces, and plugins
- Plugin frameworks should rely on interfaces and config to stay approachable
- Many interfaces are configurable too
- We need to invest in MFE interfaces and frontend plugins

Thank you!

Q&A