Pattern Detection

Arek Kita | Software Engineer & Computer Scientist

Agenda

- BC Intro
- Why Pattern Detector is needed?
- When it can be helpful?
- How it works? The idea
- Quick demo
- Next steps and your engagement
- Q&A

Being Backward Compatible

We have Backward Compatibility policy **now in place**! ...so how we make now any **major** changes?

Being Backward Compatible

We have Backward Compatibility policy **now in place**! ...so how we make now any **major** changes?

- There are different patterns we can use to maintain compatibility
  - `feature toggles`
  - component modes (`compat` vs `adaptive`)
  - `strategies` (old API is one of the impl strategy of new API)
  - `proxies` and `adapters`
  - `profiles`
  - Action for *all devs*: gather them as **Best Practices**
- **Last resort**: *unavoidable* breaking change forces migration of old code to `compat package`

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?
    - Partners can customize almost everything in AEM
    - There is no `internal`, `final`, etc

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?
  - Partners can customize almost everything in AEM
  - There is no `internal`, `final`, etc
  - Who reads AEM documentation? 100% of customers? Really?

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?
  - Partners can customize almost everything in AEM
  - There is no `internal`, `final`, etc
  - Who reads AEM documentation? 100% of customers? Really?
- A **need** to inform customers *instantly* about our API surface

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?
  - Partners can customize almost everything in AEM
  - There is no `internal`, `final`, etc
  - Who reads AEM documentation? 100% of customers? Really?
- A **need** to inform customers *instantly* about our API surface
  - Customers *want* to be prepared for upgrades and *possibly avoid compat mode*

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?
  - Partners can customize almost everything in AEM
  - There is no `internal`, `final`, etc
  - Who reads AEM documentation? 100% of customers? Really?
- A **need** to inform customers *instantly* about our API surface
  - Customers *want* to be prepared for upgrades and *possibly avoid compat mode*
  - What about `health check` that informs about *misuse* on customer instance?
  - To be checked a long before upgrade...

Internal vs public - why pattern detection?

Many of our **major** changes might be related to Adobe *internal components*

- so why `compat package` needed for those?
  - Partners can customize almost everything in AEM
  - There is no `internal`, `final`, etc
  - Who reads AEM documentation? 100% of customers? Really?
- A **need** to inform customers *instantly* about our API surface
  - Customers *want* to be prepared for upgrades and *possibly avoid compat mode*
  - What about `health check` that informs about *misuse* on customer instance?
  - To be checked a long before upgrade...

What is **unsafe** to be used, changed or customized?

Use cases for PD

1. `Upgrade` project assessment
    - What will happen after upgrade? (prediction)
    - How much upgrade tech debt I have?

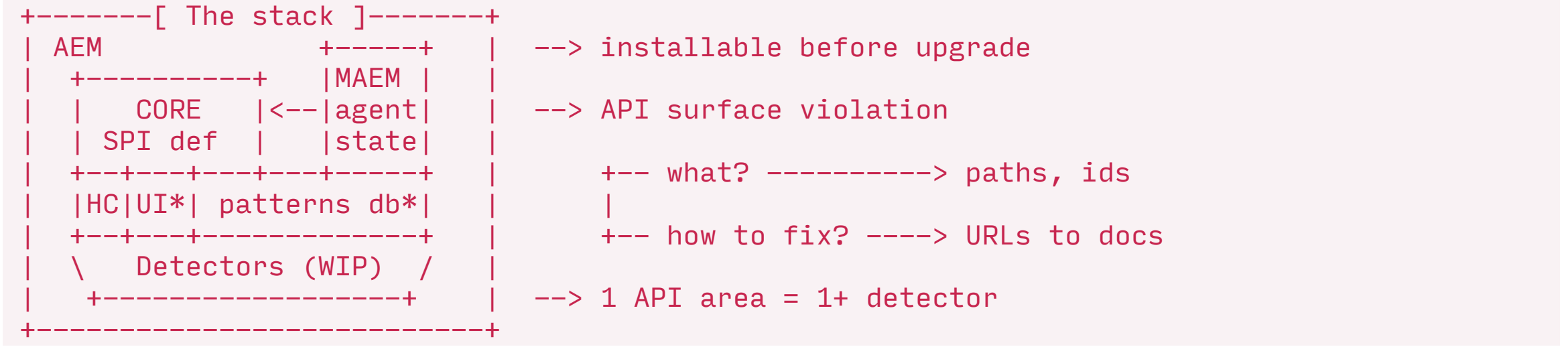
Use cases for PD

1. `Upgrade` project assessment
    - What will happen after upgrade? (prediction)
    - How much upgrade tech debt I have?
2. Tool for `development`
    - Am I allowed to use/change AEM component?
    - Will I affect future upgrades, SPs, CFPs?
    - Quick feedback cycle (less costs)

*) PD = Pattern Detector

What is unsafe? — know how

That's a goal of *pattern detection*.



* = not impl yet; MAEM = Mapping AEM initiative (please see more info)

- AEM automated comprehension system **but PD uses *only* MAEM agent**
- Friends: Apache Spark + Apache Zeppelin (outside of AEM and beyond PD)


DEMO TIME!

Major violations (as of now)

1. **Overlays** by path
   - something that is not coming from `content` or `settings` (components)
   - *solution*: requires compat switch to be **on**
     - UIs will be rerouted to their legacy versions
     - Java logic in legacy mode
2. **Overrides** via supertype inheritance
   - still no list of `final` components
   - *solution*: compat package installed but *not necessarily* to stay **on**
3. **Overwrites** of `/libs`
   - based on vault packages
   - *solution*: **unsupported case** (there are better alternatives)
4. Relying on **internal OSGi imports**
   - `0.0.0` exports for Oak (actually no version exported at all)
   - *solution*: **unsupported case** (only uber-jar API should be used)

Demo summary

- Simple! We don't want as complex analyzer as the whole AEM!
  - No possibility to detect everything => the most important areas!
  - No external *BigData* systems (limited comparing to MAEM)
- Communication tool between Adobe and developers
  - a good extension of static documentation @ `docs.adobe.com`
  - Links or IDs to Adobe public **KB with best practices** would be the best!
- Customer receives a report
  - **goal**: able to monitor compatibility progress before and after an upgrade
- Simple health check implemented
- Available as Felix inventory
  - it should be a part of ZIP configuration status
- Uses MAEM agent under the hood to get the AEM state
- no UI or dashboard yet

The current state / Next steps

- DONE: POC created
  - allows to find wrong overlays (not related to content)

The current state / Next steps

- DONE: POC created
  - allows to find wrong overlays (not related to content)
- TODO: Implement more detectors to cover more areas
  - based on MAEM agent
  - Java API
  - clientlibs
  - OSGi configs (internal, deprecated)

The current state / Next steps

- DONE: POC created
  - allows to find wrong overlays (not related to content)
- TODO: Implement more detectors to cover more areas
  - based on MAEM agent
  - Java API
  - clientlibs
  - OSGi configs (internal, deprecated)
- TODO: Create an *initial* **Pattern Database** based on input from teams
  - contribution by teams

The current state / Next steps

- DONE: POC created
  - allows to find wrong overlays (not related to content)
- TODO: Implement more detectors to cover more areas
  - based on MAEM agent
  - Java API
  - clientlibs
  - OSGi configs (internal, deprecated)
- TODO: Create an *initial* **Pattern Database** based on input from teams
  - contribution by teams
- TODO: Make it **product**(ion) ready for 6.4
  - 6.4: OOTB
  - <6.1, 6.3>: installable as additional package

Next steps (proposal)

- `TODO`: Asses which patterns could be inlined as AEM metadata, i.e.:
  - `final`, `private`, `deprecated` properties for UI components
  - `@Deprecated`, `@Internal` class, methods usages
  - bundles with `internal` category
  - Slowly merge pattern database into AEM itself!

Next steps (proposal)

- `TODO`: Asses which patterns could be inlined as AEM metadata, i.e.:
  - `final`, `private`, `deprecated` properties for UI components
  - `@Deprecated`, `@Internal` class, methods usages
  - bundles with `internal` category
  - Slowly merge pattern database into AEM itself!
- `TODO`: To be used internally in Evergreen pipeline (+ Mockingjay)
  - depends on quality of patterns

Next steps (proposal)

- `TODO`: Asses which patterns could be inlined as AEM metadata, i.e.:
  - `final`, `private`, `deprecated` properties for UI components
  - `@Deprecated`, `@Internal` class, methods usages
  - bundles with `internal` category
  - Slowly merge pattern database into AEM itself!
- `TODO`: To be used internally in Evergreen pipeline (+ Mockingjay)
  - depends on quality of patterns
- `TODO`: UI dashboard for customers where violations can be mark as false positives etc

Next steps (proposal)

- `TODO`: Asses which patterns could be inlined as AEM metadata, i.e.:
  - `final`, `private`, `deprecated` properties for UI components
  - `@Deprecated`, `@Internal` class, methods usages
  - bundles with `internal` category
  - Slowly merge pattern database into AEM itself!
- `TODO`: To be used internally in Evergreen pipeline (+ Mockingjay)
  - depends on quality of patterns
- `TODO`: UI dashboard for customers where violations can be mark as false positives etc
- `TODO`: Measuring detection rate as KPIs (in Mockingjay)
  - false suspicions
  - cases not detected **but** harmful

Bottom line

Collect anti-patterns here! *[WARN]*

Bottom line

Collect anti-patterns here! *[WARN]*

Propose new AEM areas to be analysed!

Bottom line

Collect anti-patterns here! *[WARN]*

Propose new AEM areas to be analysed!

Collect interesting cases for detection! *[INFO]*

Bottom line

Collect anti-patterns here! *[WARN]*

Propose new AEM areas to be analysed!

Collect interesting cases for detection! *[INFO]*

Watch for MAEM reports! Spot the trends!


THX! Q&A
wiki | scm