Characterizing outdateness with technical lag

Jesus M. Gonzalez-Barahona

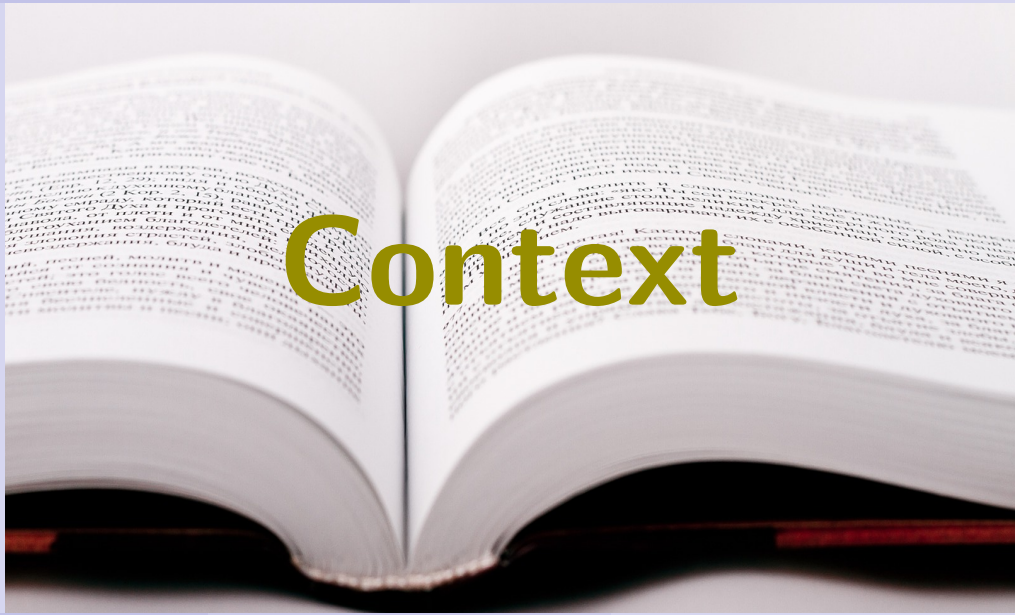
Universidad Rey Juan Carlos

@jgbarah https://jgbarah.github.io/presentations

3rd Intl. Workshop on Software Health (SoHeal 2020) July 3rd 2020

The plan

1. Context
2. Outdateness
3. Outdateness as technical lag
4. Applications of the model
5. Final notes

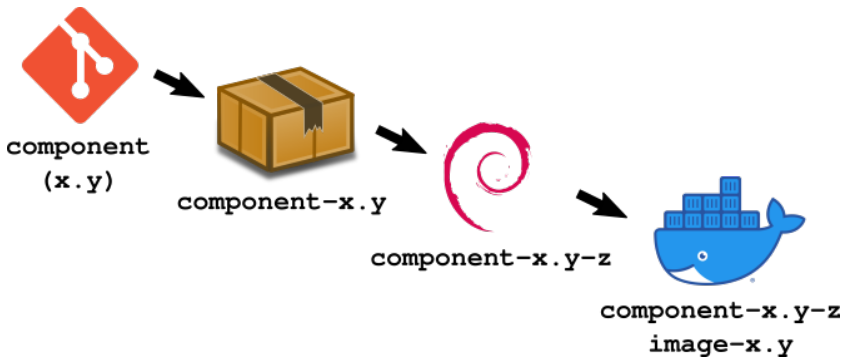


Context

Context

Applications are composed of tens, maybe hundreds of components

Each component is normally used as a package ...and that package has a story


component
(x.y)
component-x.y
component-x.y-z
component-x.y-z
image-x.y

Main goal:

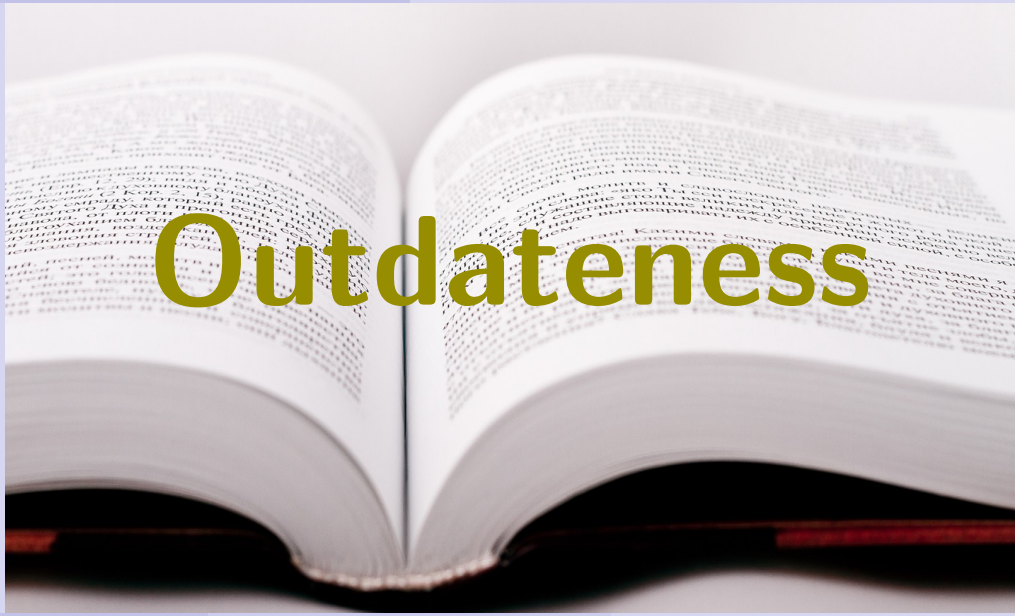
How can we compute outdateness
for an application
considering all its components?

Secondary goals:

- Metrics useful for several situations
- Factors imposing a lower bound on outdateness
- Metrics for characterizing an ecosystem

How:

Technical lag framework



Outdateness

Outdateness of an application

How outdated it is,
due to its components being outdated?

How old are its components
with respect to the latest version
available from upstream?

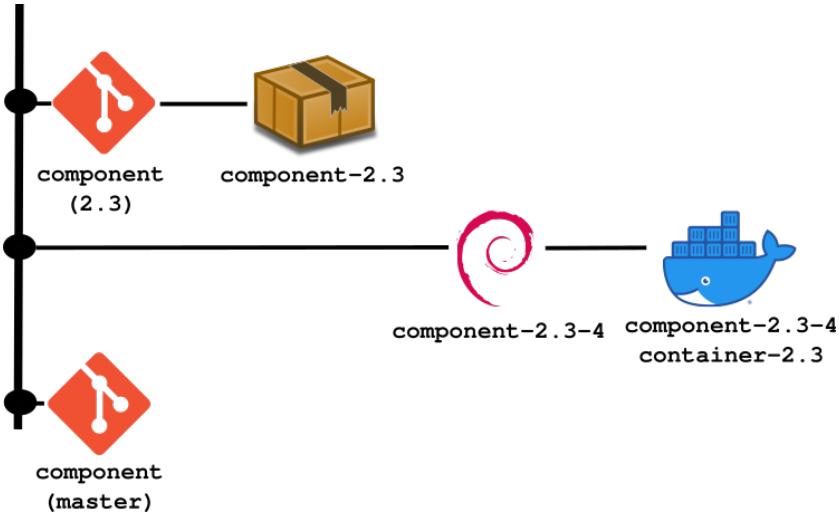
Idea

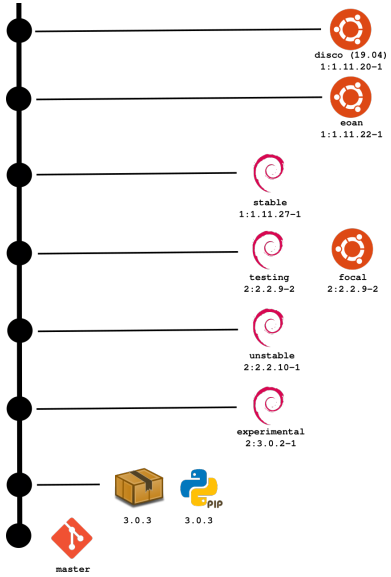
Most up to date:

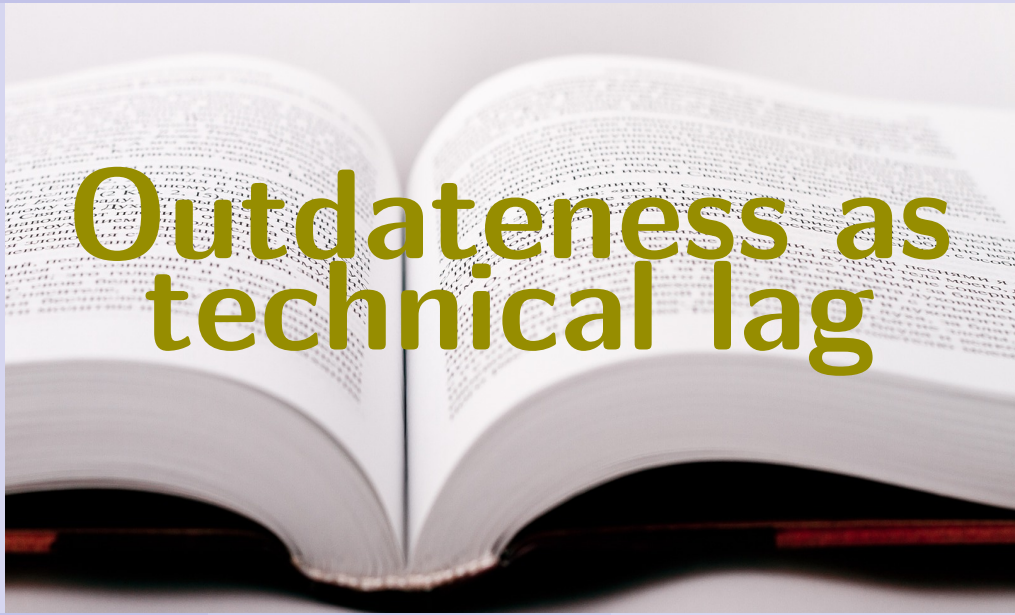
current master in upstream git repo

Map all packages to git commits

y


component
(2.3)
component-2.3
component-2.3-4
component-2.3-4
container-2.3
component
(master)





Outdateness as technical lag

Technical lag

$\mathcal{F} = (\mathcal{C}, \mathcal{L}, \textbf{ideal}, \textbf{delta}, \textbf{agg})$

- $\mathcal{C}$ set of component releases
- $\mathcal{L}$ set of possible lag values
- **ideal** : $\mathcal{C} \rightarrow \mathcal{C}$ function returning the "most preferred" component release
- **delta** : $\mathcal{C} \times \mathcal{C} \rightarrow \mathcal{L}$ function computing the difference between two component releases
- **agg** : $\mathbb{P}(\mathcal{L}) \rightarrow \mathcal{L}$ function aggregating lag values for a set of components.

Computing difference

Difference between two components
is the difference
between their two most likely commits
in the upstream repo

Outdateness

Technical lag
(with previous definition for difference)
between a component
and current upstream master

Component:

$$\textbf{outdateness}(C_{iv}) = \textbf{techlag}_{\mathcal{F}_A}(C_{iv}) = \textbf{delta}(C_{iv}, \textbf{ideal}(C_{iv})) = \textbf{delta}(C_{iv}, C_{recent})$$

Aggregation:

$$\textbf{outdateness}(A) = \textbf{agg}(techlag(C_i) C_i \in \textbf{components}(A))$$

Applications of the model

Minimum outdateness

- Minimum outdateness possible is outdateness of latest package by upstream
- Influenced by publication practices
- Different collections, different minimum outdateness for same component
- Example: Django

$$Pypi < Debian_{testing} < Ubuntu_{focal} < Debian_{stable}$$

Collections

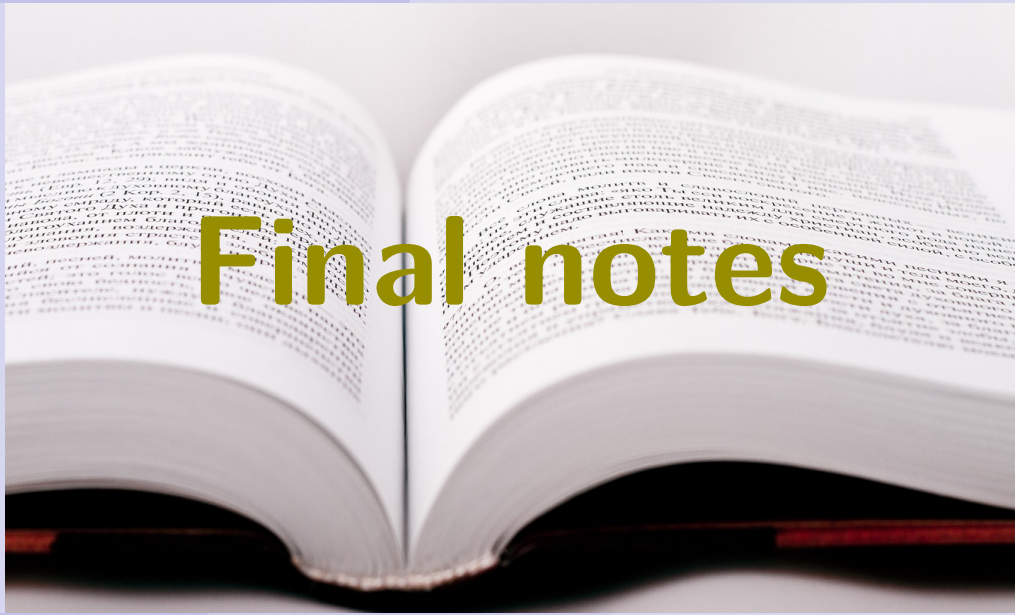
- Characterizable by mean / median outdateness
- Example: LTS, testing, experimental releases
- Example: components from Pypi or from Debian
- Example: effect of pinning dependencies

Applications

- For an application, mean / median outdateness can be computed per collection (constrained by dependencies)
- Decisions on dependencies for a certain collection

Comparing collections

- Absolute outdateness
- Dependency outdateness



Final notes

Conclusions

Technical lag can be used to precisely define outdateness

Outdateness can be used to:

- compute impact of release policies
- compare collections
- compute the effect of constraints on dependencies
- make decisions on dependencies, collections

Credits

Book, by NikolayFrolochkin, Pixabay.
License: Creative Commons CC0