Default Retirement Dates

Table of Contents

Document Info

Attribute	Value
Status	DRAFT v0.1 (2020-08-24)
Target Version	1.27
Lead	Dave Watkins

Overview

This design doc discusses approaches to implementing default retirement dates in Waltz. The driver behind this enhancement is to ensure data consistency in Waltz.

Waltz captures retirement dates at both the application level (*planned* / *actual* app retirement) and at the application/measurable level (.e.g. the retirement date for a function within an app).

Inevitably these data sets can end up with divergence. This design doc seeks to clarify the situation when these divergences are permissible and when they should be prevented.

The table below outlines the various combinations of these two dates and states if they are valid or invalid. Date in the table are represented with the sequence: `t0`, `t1`, `t2`. Earlier dates have smaller numeric components.

Table 1. State rules

Id	App planned retirement date	App/measurable retirement date	Permissible	Reason
`r1`	`t1`	`t1`	Ok	App/measurable date is the same as app retirement
`r2`	`t1`	`t0`	Ok	App/measurable date is before app retirement
`r3`	-	`t1`	Ok	App is simply removing a measurable, not retiring in it's entirety
`r4`	`t1`	`t2`	Bad	App/measurable date is after app retirement. App/measurable date will become `T1`.
`r5`	`t1`	-	Bad	App/measurable is not marked for retirement despite app having a retirement date.

Proposal

Core Waltz changes

Waltz should indicate that an app/measurable retirement date is in breach (later) than the overall app retirement date and only allow users to edit the date to an equal or earlier date (`r4`). Waltz currently does not support user entry of the overall retirement date and therefore `r5` only concerns batch loading (see below).

(Optional) Client-site batch job changes

Client installations may need to modify jobs to support rules `r4` and `r5` to ensure consistency when retirement date is set via a batch process.

Appendices

Appendix A: Original ask from Architects

Below is a (lightly edited) copy of the request. It is included here as it provides the driving rationale behind the design options outlined in the main sections above.

Disinvest Date:

- If `system-of-record` has a disinvest date, Architect can set a more aggressive date i.e. `Waltz date < system-of-record` date as an override
- If `system-of-record` has no disinvest date, but Waltz function is disinvest, Architect can set a disinvest date in Waltz as an override

Disinvest Date:

- previous `system-of-record` disinvest date equals Waltz disinvest date, then no overrides exist and `system-of-record` date is synced to Waltz
- If previous system-of-record disinvest date is not equal to Waltz disinvest date, then overrides exist:
    - If `system-of-record Date > Waltz Date`, override is preserved and date is **NOT** synced.
    - If `system-of-record Date ⇐ Waltz Date`, override is removed and dates are synced from now onwards.