Operationalisation: development

A CI/CD pipeline with GitHub

Basic elements of a pipeline:

1. Version control protocol
2. Collaboration and documentation protocol
    a. GH issues, wiki documentation
3. Structured branching strategy
    a. Git Flow or GitHub Flow
4. CI
    a. Automated builds and tests after code is merged into a central repository (source)
5. CD
    a. Changes released automatically to users after passing a series of predefined tests (source)

GitHub Issues

Source: documentation
Source: video

- Also used extensively by `schema.org` community

Workflow:

1. Navigate to the issues page and create a new issue
    a. Assign specific users to the task and add tags in the right-hand panel
2. Work on the issue and commit changes
    a. A commit message containing "#<issue-number>" will link that commit in the issue's dashboard
    b. The commit message "... fixes #<issue-number>" will automatically close the issue (fixes is a keyword in this context)

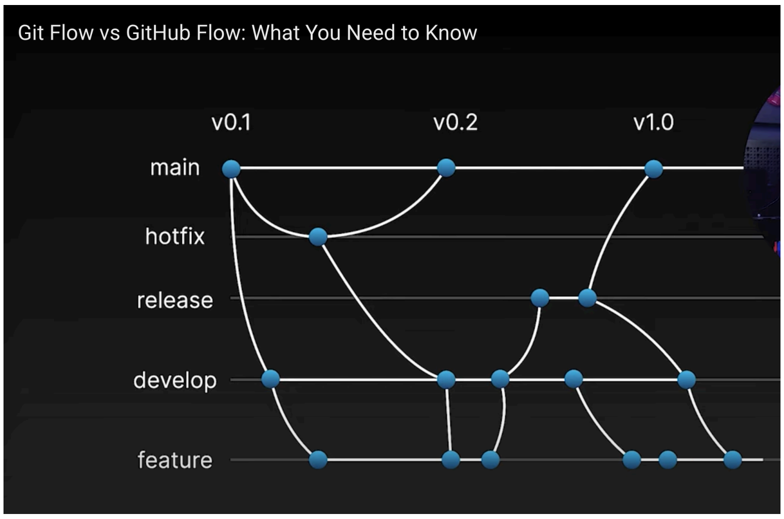
GitHub Projects

- Easy way to keep track of tasks and issues relating to a project and their status: "connect your planning directly to the work your teams are doing"
- Adaptable spreadsheet that interacts with issues and pull requests
    - Good overview/summary for contributors
- Keep the planning where the code is

Branching strategies

Source: Git Flow vs GitHub Flow

- Git Flow: designed around projects that have a strict release cycle



- GitHub Flow: optimised for teams that need to deploy continuously; more suitable for our current application
- Summary:
    - `main`: ready for deployment
    - `feature`: development happens here
- Relies heavily on automated testing to ensure that whatever is on `main` is deployable

Semantic versioning

Source

Summary

- Each version is of the form `MAJOR.MINOR.PATCH`
- Requirement: declaration of a public API ("MUST")
- Contents of a released version may not be modified
- Increment:
    - MAJOR: when making a backwards-incompatible API change
    - MINOR: when adding new features in a backwards-compatible way
    - PATCH: when making backwards-compatible bug fixes

Pre-release versions

- May denote a pre-release version by adding a hyphen and a series of dot-separated identifiers following the patch version
    - e.g. 1.0.0-alpha

Major version 0 (i.e. version 0.y.z)

- For initial, rapid development
- Anything may change at any time and the public API should not be considered stable
- Start with 0.1.0 and increment from there

Major version 1 (version 1.y.z)

- 1.0.0 defines the public API
- When to release major version 1:

    - If software is already being used in production
    - If there already exists a stable API on which users depend
    - If backwards compatibility has become a concern, “you should probably already be 1.0.0”

A workflow for aia-o

What do we need?

1. A predictable way to integrate changes to the wiki with the existing codebase
2. A way for new potential collaborators to quickly gauge where they can contribute
3. Well-structured conventions for releases, versioning, and branching

Suggestions

- Ontology-level
    - Add a “last-edited” column to the table on the wiki
    - After each meeting, open an issue on GH detailing conceptual changes to the ontology
    - Compile issues in a GH project to allow monitoring of status
    - Branching convention: only one level deep
        - Scheduled releases: not every edit to the .owl file is reflected to users. Also following schema.org’s example.

Operationalisation: use

Creating documentation: pyLODE

- “An OWL ontology documentation tool using Python and templating, based on LODE.”
- **Documentation:** what does pyLODE understand?
- **Installation:** `pip3 install pylode`
- **Generate file:** `pylode -o path/to/output-file.html path/to/input-file`
    - Input file can be .owl or .ttl

- Requirement: include
```
<http://purl.org/aiaontology>
    rdf:type owl:Ontology ;
    dcterms:title "AIA Ontology" ;
    dcterms:description "An example ontology for AIA." .
```
  at beginning of file; used to generate metadata field in .html field

Hosting documentation: netlify

- https://aiadocs.netlify.app/
- Got a 404 on first build. Fix was: rename “home” page file to index.html; set Publish directory to “/” (Site configuration > Build & deploy > Build settings)

Recommendations and next steps

Operationalisation depends to a great extent on the *ease of use*

1. Organise website (https://aartum.io) + “prettify”
2. Structure releases: not every change to ontology on GitHub should immediately reflect to users
    a. Branching protocol - whatever is on main is always deployment-ready
3. Introduction video + documentation (10 minute read)
    a. What is aia-o? A standardised ontology for describing the impact humans have on their environments
    b. Why do we care about an ontology for anthropogenic impact accounting? To standardise quantification of environmental impact projects
    c. Why do we care about that? Quantify communal benefits => aartum
4. ‘Tutorial’
    a. Alex already has “How to AIAO” on wiki