Shared software development practices between UCO and CASE

The purpose of this document is to describe policies related to versioning and developing the UCO and CASE ontologies, through the lens of software development. The scope of this document encompasses revisions of Git-tracked files, especially at the ontologies' repositories [1,2]. This document does not describe decision processes for what changes to apply to either ontology; instead, that topic is left to the respective communities' Operations Guide(s) [8,9].

Change Register

Approval Date	Description of Change
Dec 2, 2019	Initial Publication

References

[1] https://github.com/ucoProject/UCO
[2] https://github.com/casework/CASE
[3] https://tools.ietf.org/html/rfc2119
[4] https://semver.org/
[5] https://nvie.com/posts/a-successful-git-branching-model/
[6] https://www.w3.org/TR/owl2-syntax/#Versioning_of_OWL_2_Ontologies
[7] CASE Community Bylaws: https://caseontology.org/resources/bylaws.html
[8] UCO Community SOP: https://docs.google.com/document/d/1f4DE6vJD88QBFzA4sORjWowRiSk_A746Q1UtNT159NI/edit?usp=sharing
[9] CASE Ontology Committee Operations Guide: https://docs.google.com/document/d/15I8OMt9P33C3nIbeyr4Euka2dyOoklO2-mj97Kw30o4/edit?usp=sharing
[10] CASE Community and UCO Community Agreements:
FUTURE: Memorandum of Agreement Between CASE and UCO Community Governance Committees

Requirements nomenclature

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [3].

Versioning

To clearly define the stability of the ontologies to adopting tools, sub-ontologies (e.g. as CASE relates to UCO), and investigative organizations, UCO and CASE SHALL follow Semantic Versioning ("SemVer") 2.0.0 [4]. The version of SemVer is selected by merit of it being the version on November 25, 2019 when the initial version of this document was published. Principally, the versioning practice is to specify the social (but not legal) contract laid out on the semver.org home page:

> """
> *Given a version number MAJOR.MINOR.PATCH, increment the:*
>
> *MAJOR version when you make incompatible [ontology] changes,*
> *MINOR version when you add functionality in a backwards-compatible manner, and*
> *PATCH version when you make backwards-compatible bug fixes.*
> """
> ***Note***: The word “API” was replaced above with the word “[ontology]”.

The semver.org home page also describes decision points for deciding which portion of the version vector must be updated when introducing a change. For instance, deprecating a feature should trigger a minor version increment.

Ontology IRIs and Ontology Version IRIs

Both the UCO and CASE communities have specified their Ontology IRIs. No commitments have been made at the time of this document version to make the Ontology IRIs resolvable on the web. Consideration SHOULD be given to such a commitment upon the official release of a major version.

No commitment to supporting Ontology Version IRIs has been made by either community. If such a commitment is made by either community adherence to W3C recommendations SHOULD be followed. See [6]:
https://www.w3.org/TR/owl2-syntax/#Versioning_of_OWL_2_Ontologies

Prerelease versioning

UCO and CASE MAY follow the SemVer practice of designating prerelease versions, using a hyphen after the required major-minor-patch triple. The communities agree to limit prerelease version declarations to numbered "beta" releases, starting with "beta.1". A version string for such a release would be "X.Y.Z-beta.1". This is in recognition that developers interested in "alpha"-quality releases have the flexibility provided by Git to track a specific commit in the development history. Meanwhile, ontology developers or adopters may want the breakage-avoidance that "beta" periods provide. To prevent adoption backups from prolonged "Beta" periods, a limit on beta periods can ensure development and community progress.

A beta period SHOULD include a declared notice of its duration, indicating the appropriate time given the context of the beta.

Versioning and supplementary data

Management and versioning of supplementary data (sample data, documentation, etc.) will be handled separately from management and versioning of the ontology. Versions of supplementary data will include explicit specification of the applicable ontology version.

Version, support, and release schedule

The Presiding Directors of the UCO and CASE communities will collaborate on version, support, and release schedules.

Branching

The UCO and CASE communities use a Git branching model that clearly separates the "Stable" public API version from current development. The branching model is adopted from Vincent

Driessen [5], and in this document is referred to as "The Driessen Model." Particular to CASE and UCO, the Git branches are:

- "master" - The current public ontology. The only commits that happen on the "master" branch MUST be merges of "release" branches.
- "develop" - This can be considered the "nightly API". As in the Driessen model, the commits that occur in this branch SHALL be non-fast-forwarding merges of release branches, hotfix branches, and feature branches.
- "release-$version" - A branch created in preparation to declare a version. Particular to UCO and CASE, if a prerelease is to be declared, the initial commit making a release branch SHALL increment version descriptions to their initial "beta" designation (e.g. "release-1.2.0-beta.1"), which would be dropped at the conclusion of the beta period. Updates to release branches otherwise follow the Driessen model.
- "hotfix-*" - As in the Driessen model. Started from "master" branch only.
- Tags - A Git tag (not branch) of the "master" branch, at the commit where a release branch has been merged in.
- "next-major" - This branch is not specified in the Driessen model. To prepare for likely-distant "major" version releases (such as for removing a deprecated feature), the UCO and CASE communities SHOULD use a branch "next-major" that merges feature branches containing backward-incompatible changes with the current major version. Such a branch MAY make frequent merges of "develop" to ease major-version transitions.
- Any other branch name - Called "Feature branches" in the Driessen model, these branches are where ontology development will occur. These branches SHOULD merge any since-branch-point updates to the "develop" head (e.g. to confirm unit tests still pass) before being presented for Pull Requests. (That is, the branch "develop" SHOULD be merged into the feature branch; the branch should be tested; and then a Pull Request should be presented.) To plan for changes for next major releases, feature branches MAY branch from "next-major" instead of "develop".

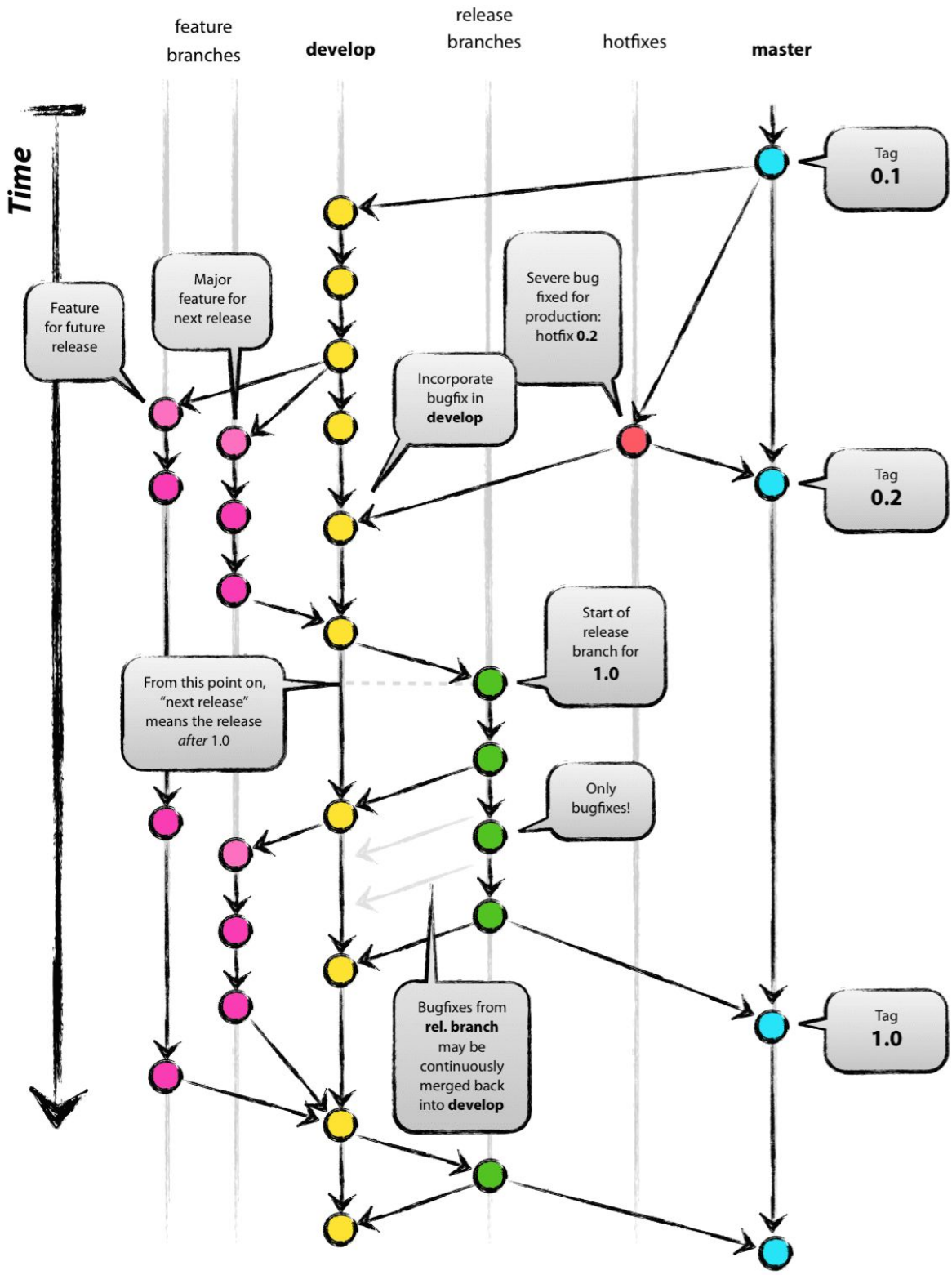

Figure 1: The Git branching model, illustrated by Driessen [5]. Ontology developers would test development tool versions with development CASE branches (pink, on the left). Adopters' production tool versions would look only at the master branch (blue, on the right).

Logging changes

To illustrate activity since prior releases, the UCO and CASE communities SHALL keep a file at the top of their respective ontology repositories, named "ChangeLog." Updates to this file SHOULD be included among the changes in a pull request to the development branch (that branch being "develop" or "next-major"). Lines in this log SHOULD be short and simple descriptions of the contributions of each merged feature branch, similar to the title line of a pull request. Fuller descriptions of the pull request, or links to formal documentation of the changes in the pull request, SHOULD be left in the Git commits that merge the feature branch, not the change log.