Product maintenance

The goal of this document is to discuss how to focus the resources we have on product maintenance within our current resources. Resources are partially core team - the core team has some generic funding through partnership fees & membership fees that is not used on other key tasks (supporting events, attending sprints, partner liaison, running the servers, running the continuous integration suite, packaging releases, providing support, technical mentoring, engaging in dev discussions etc). These are supplemented by volunteers and shops providing pro bono work. The focus here is not on growing these resources but respecting them in order to not lose them is a focus.

Open source

CiviCRM is an open source product. This is obvious but has to be stated as the reason for using it for many is the ability to contribute & evolve the product in ways they find useful. On one hand this is great because it means the areas that people are prepared to invest in improve and areas people care about less get let attention. It means the development is tied to real world needs & uses.

It does provide challenges, however. In general partners prefer to fund their own developers to do fix bugs, rather than contributing to a larger bug-fix pool of funds. However, these fixes require (generally unfunded) time to review. Often the reviewer spends more time on the review than they would have on fixing the bug themselves, and not infrequently more time than the original submitter spent on getting it into the PR queue.

There is no ‘fix’ for the tensions involved here. It’s just a case of acknowledging that the work done to support contributors is a key part of maintaining an open source product, but that the resources required to do that exceed those available.

What does product maintenance entail?

The ultimate goal is to ship a product that is free of bugs and regressions, while supporting the efforts of contributors to improve the product.

(There are also some strategic goals for CiviCRM's evolution that fall outside product maintenance. For example the core team has spent over 100 hours on drupal 8 development & resolving WordPress issues for reasons that fall outside the product maintenance part of their work).

The specific tasks in product maintenance are

- Identifying bugs (aka triage)
- Fixing bugs
- Reviewing contributed work
- Reviewing proposals for contributed work

It's worth noting that each of the last 3 of these could, by itself, suck up the entire core team budget (not just the part of it available for product-maintenance) so the goal here is to prioritise & using gitlab to do triage well is key to managing the prioritisation.

Given that there are not resources to fix all bugs the proposal is that the focus is on (these are in order but the review prioritisation is more nuanced than the below)

1) Resolving all critical bugs
2) Resolving all recent regressions 'straight away'.
3) Reviewing all contributed PRs that have unit tests in the PR queue
4) Reviewing all contributed bug fixes in the PR queue
5) Reviewing other contributed work in the PR queue & reviewing proposals

Identifying bugs - aka triage

Content now here
https://github.com/civicrm/civicrm-dev-docs/issues/531

Fixing bugs

Not much to say here as this is something we have long focussed on, perhaps to the detriment of some other areas of produce maintenance.

The bug fix focus for product maintenance is on critical bugs & regressions followed by supporting community efforts to fix bugs. I propose we add a tag to gitlab for 'bug' to help prioritise those for review.

Reviewing contributed work

Reviewing contributions essentially means looking at the PR queue. As at writing there are 149 pull requests in the queue. This probably equates to upwards of 300 hours of work that has already been done and upwards of 300 hours of work to get them all reviewed. Most of the work done was probably funded but perhaps as little as 0 of the hours to get them reviewed is funded.

This resource gap is not easily solved so we need to think about efficiency within it. A specific issue is that it can take as long to find a PR that is reviewable in a queue with ~150 PRs as to actually review them.

Here are some principles I propose

1. Recognise our reviewer resources are our least funded and scarcest resources. We need to be respectful of them.
    - Not only is review work generally unfunded, it is time consuming and it involves taking on responsibility for a change that is not to their benefit but they will sure have to ensure is fixed if it causes a problem, and potentially bear the brunt of any criticism from the community.
    - Reviewers will make mistakes and we need to accept that in a way that does not reduce the number of people prepared to do review.
2. The PR queue should be a place for PRs currently active or fully reviewable.
    - PRs suck up time just by existing. There are a number of PRs in the q which I have spent maybe 6 minutes looking at & concluded that I wouldn't review it because I would never merge it without a test & there is not one. The problem is I might have spent those 6 minutes 20 times over the lifetime of a PR
3. A PR does not have to be open for discussion to take place or for it to be accessible later.
    - Where a PR cannot be resolved easily we should consider the primary place to track it as being gitlab & ensure it is linked to an issue there. If the PR is closed it & the discussion on it can be found from gitlab.
4. We should close PRs that are not ready for review
    - if we look at a PR & think 'I wouldn't merge this without a unit test' we should make that comment on the PR and close the PR with a request to re-open in when there is a test
    - If the PR is not truly recent & has become a discussion piece we should close it with a request that it is re-opened when the discussion has resolved.
    - If a PR is stale we should try close it & ask for it to be re-opened
    - If the quality of code is not really there we should consider closing & doing any mentorship offline
    - If the approach is not right we should consider closing & requesting a new PR with a new approach

- If a PR is truly active we should hold off (new PRs often generate discussion & we should encourage that, but once they are no-longer near the top of the list ad hoc discussion becomes unlikely)
- The PR queue is nobody else's to-do list. Sometimes people track their own work through their own open PRs. That's not a legitimate use of the PR queue.
- WIP Prs should only stay open for a short period

5. We should be careful about reviewers writing tests for other people's PRs (or taking on fixing the bug if the PR is not up to scratch)
    - It has been frequent for reviewers to write tests for other people's PRs. If someone submits a bug fix that we would not merge without a test we have often written them. However we need to recognise that we are prioritising limited resources and consider closing with a suggestion to re-open when a test is present unless other factors. (There are numerous reasons why a reviewer might still choose to write a test but they should at least feel the option is theirs to request a test without having to leave the PR in the queue)
6. We should identify & prioritise PRs that fix bugs
7. We should identify & prioritise PRs that have unit tests
8. We should identify & prioritise PRs where 'most' of the work has been done
    - In general by the time a PR has been submitted somewhere between 10% & 90% of the work on that bug has been done. Try to identify & focus on those closer to 90% & close those closer to 10%.
9. We should identify & prioritise PRs by people who act as reviewers themselves
    - This is out of respect for the work that reviewers do, and in recognition of the fact that a lot of their PRs are in response to that work rather than their own paid work anyway.
10. We should identify & prioritise PRs that have had endorsement from community members
    - In particular this should be when the endorsement says something like 'I've tested this & it works' as they have done part of the job of the PR already.

11. We should identify & prioritise PRs by people who have a track record of engaging in review & treating reviewers with respect
    - It is not infrequent for a reviewer to find that the PR submitter feels that by submitting the PR they have done their bit and to not wish to put any more time into the PR after that point. This places an implicit value on their time and none on the reviewer's time. However, the majority of submitters DO appreciate the time reviewers put into doing review. It is valid for reviewers to focus their efforts based on return on their effort. (Note I think any discussion based on this should focus on 'reviewing Matt's PR would be a good idea since he is always really responsive. I'm sure most reviewers have a mental list of people they have been stung by, but discussing those is not helpful). In the bigger picture we need to educate people that submitting a patch for review is not job done but job started.

12. We should try to improve our review soft skills
    - We often see PRs that have had a lot of discussion, sometimes on trivial details, without big picture review. In some cases reviewers have had people change code on fairly stylistic issues without looking at big suggestions like 'can we share this code with the code in class x'. In others the problem is the actual motivation for the change is not agreed. We need to try to get better at describing the big picture of a PR in some instances. Ask ourselves 'what would it take to get this merged'

13. We should provide a way for people to pay for PR attention
    - This has been proposed as 2 hour pre-purchased blocks of core team time to review a PR. It does not involve a commitment to merge and may take the form of discussion, a substitute fix and may result in a quote for more time if needed. But, it does mean that the CT member will spend 2 hours within a week on that issue or issues. (We should set up an easy mechanism for this if people do wish to take it up).

14. We should try to give timely feedback on what might be blocking a PR
    - This is really a PR triage process. Often we are reluctant to comment on PRs because we know by doing so we may get drawn in or tick off some people.

15. We should meet weekly to triage the PR queue
    - Often it's really hard to do the above things in isolation. I propose we have a weekly meeting (or possibly 2 to accommodate timezones) to do PR queue triage.

Reviewing proposals for contributed work

Proposals for contributed work might be a minor wording change or they might be a significant change. The process here is currently frustrating in many ways

1. Shops wanting to make core changes find it hard to get feedback
2. CT and others are asked to spend a lot of time giving feedback on changes that may not progress, or may be of limited general interest
3. Any comment a CT member gives on a proposal may taken as a reason to commit significant resources to implementing it - with a proposed obligation on CT to commit resources to review and to accept it.

Nevertheless collaboration with contributors is a key open source advantage and a reason for using CiviCRM

My best proposal here is that

a) The triage process identifies items in gitlab that are seeking feedback (ie. those with the labels 'proposal') and
b) That a regular amount of time is time-boxed to provide response to these
c) If more time is wanted from CT than the time-box process allows that would be 'purchasable'
d) We get better at giving that feedback