Surpassing RfCs

How to improve the governance of large software changes

About me

- Gergő Tisza / [[User:Tgr]]
- Wikipedian since 2004; admin at Hungarian Wikipedia between 2005-2020, now interface administrator
- software developer @ WMF since 2013; worked on some features & proposals that ended up being quite controversial (MediaViewer; MP4 support on Commons)
- saw some WMF-community power struggles from the other side (I was on a chapter board when the WMF abolished the financial independence of chapters)
- heavily involved in movement strategy discussions as a volunteer; I was on the Product & Tech working group, then the recommendation writer team; now participating in the Movement Charter discussions

Why talk about this now?

- The Movement Charter is being drafted now; one of its main topics is how to make decisions that affect the whole Wikimedia movement.
- Currently movement-level decisionmaking about software changes happens either via RfCs on the largest wikis, or the WMF deciding things unilaterally. I think this is bad for everyone involved. The Movement Charter provides a vehicle for change, as long as we can agree what change is needed.

Problems with software RfCs 1. – Waste

- Software development is expensive. A major new feature can takes years to accomplish for a WMF software team; that probably costs a few million dollars.
- The RfC* approach: *first* the software gets fully implemented, and usually deployed; *then* an RfC decides whether it is wanted. If it isn’t, we just burned up a million dollars of donor money for nothing.

Yes, but:

RfCs allow the editor communities to get involved in decisionmaking on an equal footing, not just as a passive subject that gets “consulted”, with someone else choosing how to interpret the outcome of that consultation.

That is an important element of community empowerment and power sharing. Editors donating their time is the foundation on which the entire Wikimedia movement is built; they *should* have a say in how the software platform evolves.

The specific mechanism by which that power sharing happens is terrible, but it’s important to have *some* mechanism.

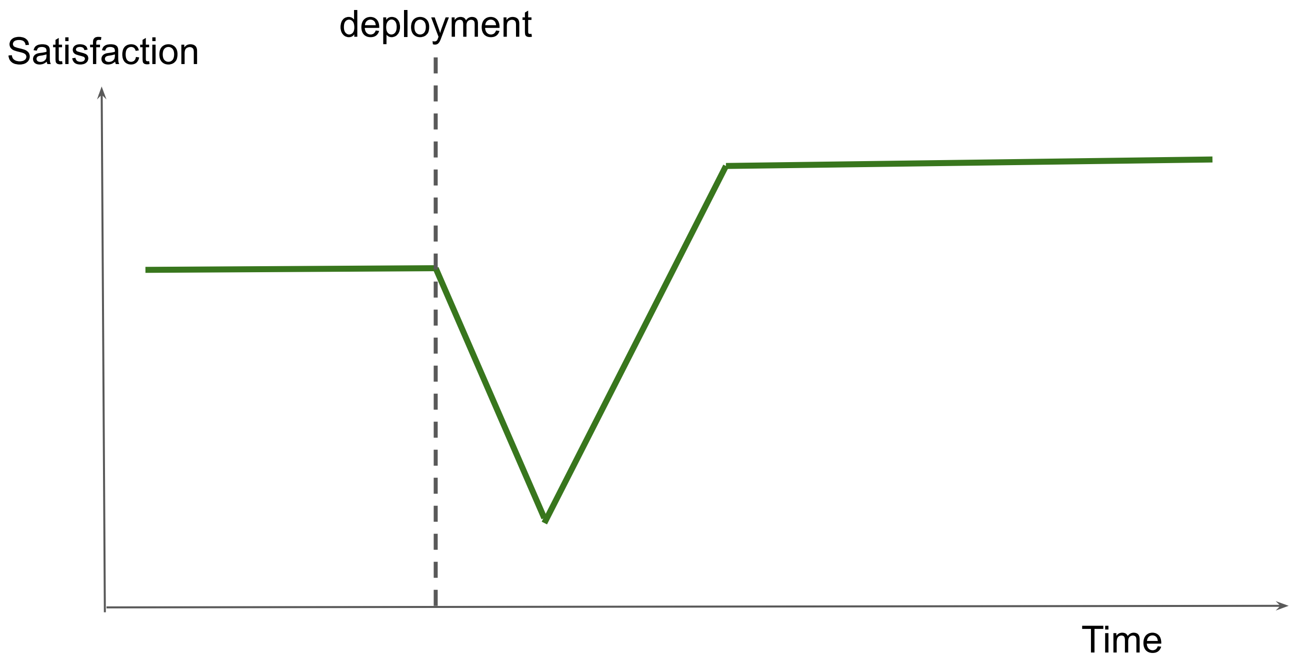
Problems 2. – Commitment and engagement

Could we just have the RfC sooner? Not really:

- There is no guarantee the outcome of an RfC will be honored. Anyone can start a new RfC to overturn an old one; sometimes within a few weeks (as we have seen with Vector 2022 on English Wikipedia).
- Way more people will want to participate when the change already happened and is affecting them than in a discussion about a future change; so it’s almost guaranteed someone *will* start a new RfC and it will have more support.

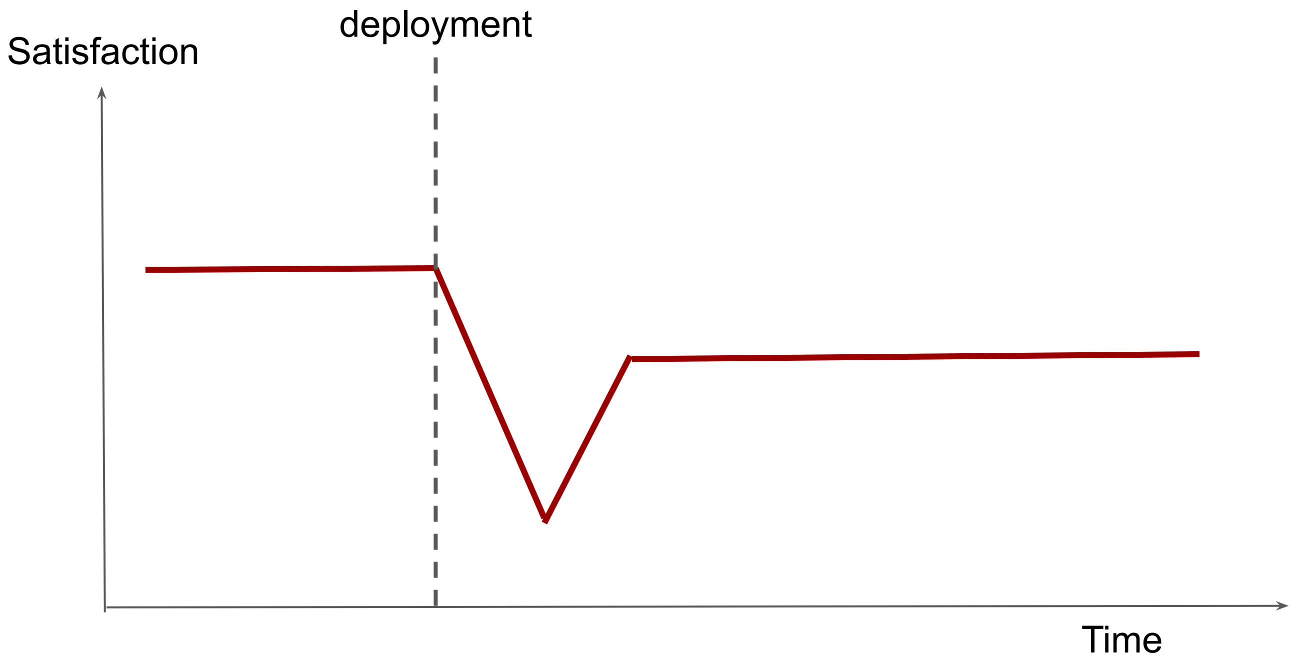
Problems 3. – Change aversion

User satisfaction when a feature change goes well:

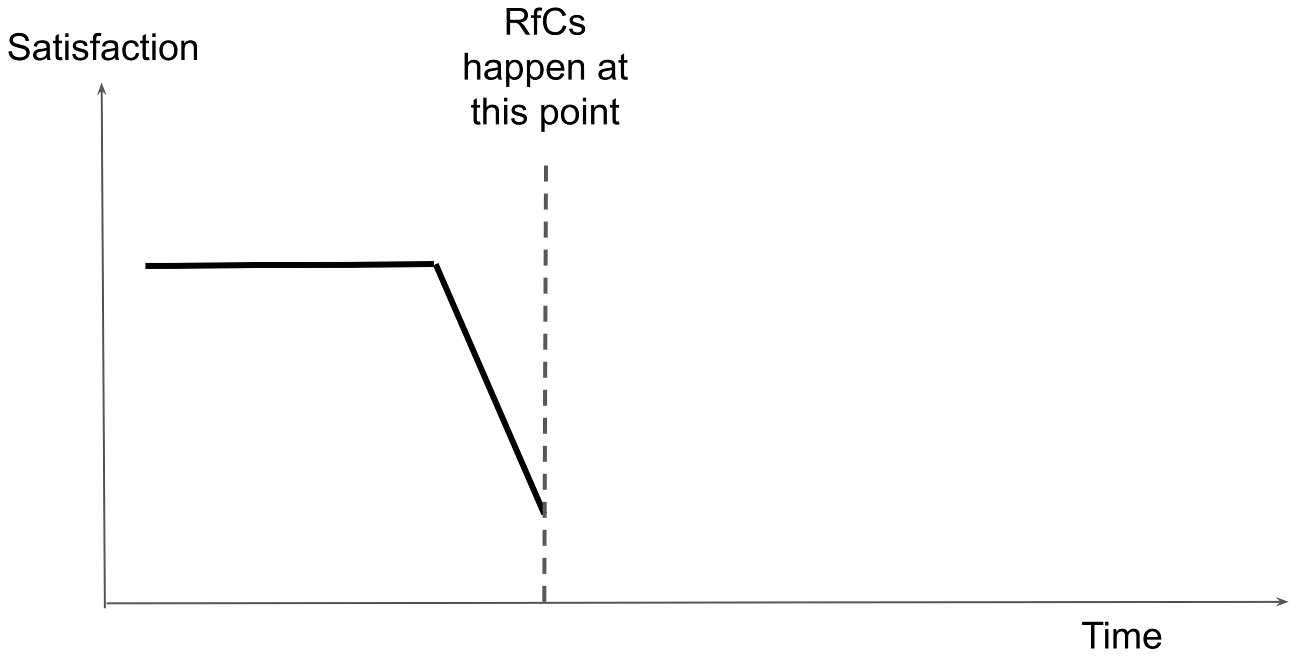


Problems 3. – Change aversion

User satisfaction when a feature change does not go well:



Problems 3. – Change aversion



Problems 4. – Quality of decisions

- Deciding what software changes are needed often requires expertise (user experience design, understanding of internet trends etc). RfCs are not good at elevating that expertise.
- Some projects use formal voting, which tends to be dominated by people who have spent very little time engaging with the issue and only have a superficial understanding of it.
- In other projects, RfCs are supposed to be decided by the weight of the arguments, but the way arguments get evaluated is fairly arbitrary and depends on who closes the RfC and how much effort they are willing to invest in it.
    - It also puts the closer in a difficult position when the majority position is not well-supported by arguments.

Problems 5. – Scale

- We have over 900 projects. There is no way for developers to keep track of, much less honor, hundreds of RfCs.
- There could be one global RfC, but then who decides the roles? Who evaluates the arguments? What language will be used for discussions?

Problems on the other side

RfCs give too much (and poorly controlled) influence over what gets *deployed*; but at the same time, editors have too little control over what gets *developed*. They can prevent things from happening, but they can’t make the things that are important to them happen.

(The Community Wishlist is a notable exception, but it represents maybe 2-3% of the movement’s software development resources.)

How would a better process look? 1. – Representation

- It needs to elevate expertise, but it also needs to ensure that editors have real power, and aren’t just passive subjects who get “consulted”, with paid staff deciding how to interpret their words. Real-world governance systems usually strike that balance via *representation*: the community elects a group that will negotiate on their behalf. The representatives (let’s call them Product Council) invest time into understanding the context of a change, and assume a role similar to that of an RfC closer. Since the Product Council is relatively stable over time, they can make longer-term commitments.

Better process 2. – Open planning process

- Editors are involved (through the Product Council) from the start; that includes involvement in deciding what gets developed, ie. prioritization between the various proposed projects.

Better process 3. – Experimentation and metrics

- Whenever possible, disputes are decided by measurement, including controlled experiments. This is difficult with RfCs due to the time scales involved and the cost of coordinating between hundreds of RfC participants, but perfectly viable with a committee.

Better process 4. – Targets

- Instead of mixing up “Do we want this thing?” and “Is it done?” in the same RfC, there are two distinct steps: first the organization and the Product Council agree that a given feature should be developed, and agree what targets need to be reached to consider it done. (This could include must-have functionality, metrics, anything that can be determined more or less objectively.)
- Then, after the feature gets developed, the Product Council (in consultation with the communities and the organization) determines whether the targets have been met and the feature is ready to be deployed (or taken out of beta, made opt-out instead of opt-in etc).