How to Have Fewer Bugs in Tiki

This page's goal is to ultimately **1- implement actionable/realistic changes of policies**, and **2- prioritize systemic tasks to improve Tiki's stability** (fewer bugs). The first thing is to have a common understanding of the problem, and then propose realistic and targeted actions that will actually help

. Then, we can measure our progress and adapt.

As of 2021-11-21, the discussions have not lead to a clear list of patterns, but many people now understand what is needed and will hopefully think about it and come back with ideas of patterns. We do have one clear and easy action item: Tiki Test Plan for new site admin but it's not enough to change the global trajectory. Until patterns are identified, this project is on hold and resources will be allocated elsewhere.

1.1. Context

Some community members express a goal to reduce the quantity and severity of bugs. Everyone agrees we want fewer bugs but how can we get there? Some of the proposals like here and here are well intentioned but if implemented

1. Would take a massive amount of effort
2. Would demonstrably cause major issues to some active community members
3. Would not have a net positive impact to reach the purported goal to reduce the number of bugs. (This is debatable of course, but there has been no demonstration of cause and effect. I am talking about a document like Versions which explains the choices/impacts.)

So some of these proposals are sure to cause pain, and will not solve the problem. So they will not be implemented.

However, the questions and concerns about bugs deserve consideration. What *would* help?

As with many things, the majority of the problems stem from a small number of root causes. Let's find the root causes and address them.

1.2. Live Chat Recordings

- Recording of related discussion at the October 2021 roundtable.
- Recording of related discussion at the November 2021 roundtable.

1.3. Four-step plan

Let's keep an open mind and seek the best possible information, avoiding patterns such as:

- https://yourlogicalfallacyis.com/anecdotal
- https://yourlogicalfallacyis.com/false-cause
- https://yourlogicalfallacyis.com/appeal-to-authority
- https://yourbias.is/belief-bias

- https://yourbias.is/the-dunning-kruger-effect
- https://yourbias.is/the-availability-heuristic
- https://yourbias.is/confirmation-bias
- https://yourbias.is/declinism

- I don't understand this section — too confusing?
    - Sometimes, some community members propose solutions which are pointless or even counter-productive. I could just ignore them, but that is not going to be very motivating for them, and it doesn't make things better over time. I am trying to educate them to try to adopt a more logical thought process. I have found these two sites to be really good, as they have helped me distinguish between some concepts I was mixing up. So I added the list of fallacies and behaviors I have seen in the Tiki community (not necessarily for this process, but in the last few years).

See also: https://en.wikipedia.org/wiki/Five_whys

1.3.1. Establish the facts

We first need to gain a common understanding: What are the patterns for the bugs?

- Is it more a concern of bug density? (Bugs/feature ratio) or bug severity? (fewer bugs but they are nasty)
- Are there really more bugs than before? Or has the density and/or severity changed? Or it's just that people report them more?
- In new features (unfinished/immature features) or in old features? (regressions)
- In base features (ex.: menu), transversal features (ex.: categories) or in specific features (ex.: quizzes)?
- Is it possible to deactivate the buggy feature?
- Is it a case of a conflict of two features (each works well independently)?
- Are the bugs difficult to fix, or easy papercut bugs?
- Intermittent or consistently reproducible?
    - For example CSRF wishes are often intermittent
- Do the bugs have error messages? Or it's that features are unusable because of bad docs, bad UI, or something else? Or it's that they randomly don't work (ex.: comments forms sometimes don't load)
- Concentrated in specific features or they are more or less evenly spread out?
    - For example, MakeToc has a lot of bugs and it's really hard to fix, because it's a flawed design. So we created AutoToc to replace it.
- In Tiki code or in external dependencies or in a weakness in how we integrate the dependency?
    - Example: WYSIWYG relies on an external project (CKEditor 4) designed for HTML and this is pretty good. However, we are also using for wiki syntax and plugins, which it's not designed for, thus, this part is more buggy (CKEditor v5 has been re-architected to address this)
- What are the use cases (Website vs Intranet vs shopping cart vs multi-lingual, etc.) with more bugs? fewer bugs?
- Happening on a Tiki community site? (tiki.org, doc.tiki.org, dev.tiki.org, etc.) or mostly in features with no community Dogfood?
- Are they issues depending on server software, configuration, version? (Windows vs Linux, Apache vs NGINX, MariaDB vs MySQL, php.ini, etc.)
- Issues related to client? Mobile vs desktop browser? browser choice, browser plugins, etc
- Issues connecting to external services? (Enterprise authentication, social media service, etc.)

- Are the bugs reported on dev.tiki.org? Do they have a show instance?
- Expectations with the release cycle
    - We expect that a release cycle with 22.0, 22.1, 22.2, etc. will get more stable with each minor release, and given that an LTS version like 21.x has a longer life, it will typically become more stable than a regular version like 22.x.
    - We expect that post-LTS releases have a lot of major changes and enhancements, and have a higher risk for bugs and regressions.
    - Now, beyond the normal expectations, are there some things that stand out? For example, post-LTS versions have way more bugs than expected, fixes are not backported far enough, backports to stable branches are causing instability, etc.
- Expectations when running from Git
    - Many active community members get Tiki from Git, and in some cases, with an automatic update. This permits to get bug fixes sooner, but it also leads to some regressions that are fixed shortly after.
    - Are they too many regressions in stable branches? (which could suggest a problematic backport process)
- Do the bugs lead to data corruption?
- Do the bugs affect unmodified Tiki instances or heavily customized instances? Given Tiki's integrated nature, we can do major refactors that work fine for unmodified Tikis upgrading, but that are problematic for Tikis with custom code. Ref.: Upstream First.

What types of bugs are we talking about? Please add in the section below.

Examples of problematic bugs

- Menu system in 18.x: Drag and drop was added in 18.0 LTS: https://doc.tiki.org/Menu#Drag_and_drop and it was very buggy. In retrospect, this should have been done in 19.x so it would have had time to stabilize before being in an LTS, or at least be optional. But we didn't expect such issues for something as simple as drag and drop. Turns out that the Tiki menus data structures is more complex than drag and drop libs can handle by default.
- CSRF security ticket system in last few versions of Tiki has too many false positives and there is no clear solution (can't turn off the feature, errors are semi-intermittent, system doesn't tell you what was the trigger so users and admins don't know how to adjust their behavior, and coders don't know what to fix): This is an improvement to address CSRF security issues in a more systematic way, instead of "whack-a-mole": CSRF Protection, but it needs more work and likely an adjustment to its design to reduce CSRF wishes.
- TableSorter: Quite a few bugs, not clear if it's upstream or our integration. But it's optional and off by default
- CkEditor WYSIWYG: Quite a few bugs, not clear if it's upstream or our integration. But it's optional and off by default
- CodeMirror: Quite a few bugs, not clear if it's upstream or our integration. But it's optional and off by default
- Sub-forums https://gitlab.com/tikiwiki/tiki/-/merge_requests/193 Added by a junior dev (no longer active)
- Tiki Parser (wiki syntax) bugs - e.g. putting a URL in parenthesis causes to include the closing parenthesis to become the part of the resulting link on the page rendered.

1.3.2. What are the root causes or common patterns?

Once we are clear on the patterns for the types of bugs, let's clarify what are the patterns for the root causes.

- **How are the bugs introduced?**
  - From Tiki coding or dependencies or change in ecosystem (ex.: move to PHP8, or Bootstrap)?
  - Introduced by junior devs, senior devs, or specific devs? It has happened in the past where a developer (now long gone) left a non-trivial amount of unfinished code in various places in Tiki. And the active devs are then stuck dealing with this. As of 2021, this has not happened in years, and we (Jonny, Marc and others) are alert to this and we will intervene if we see a risk. Also, now with Git, code can more easily be refined in a branch without affecting Tiki's released versions.
  - For regressions, so they occur when features are enhanced? Or when another bug is fixed? (suggestion a certain complexity or fragility in that part of the code)
  - Did the bug arrive through a merge operation? Ref: Semi-automatic merging period
  - Was the bug present only for a specific combination of preferences? Perhaps because the developer was testing with a different permutation of preferences?
- **When are they introduced?**
  - Are bugs introduced in trunk while the majority of the active developers are focusing on the most recent stable branch? And thus, they only really get uncovered months later once it's released and end users suffer (and by that time it's actually sometimes more work to deal with it)
  - Are bugs introduced in the last few weeks before a release?
- etc.

Once we have a clear understanding of the bugs and process, we can move to the next step

1.3.3. What can we realistically do to address the root causes?

1.3.4. How can we measure we are going in the right direction?

So we'll adapt our policies and prioririze key tasks. Now, a few months later, how do we know it worked?

- Satisfaction surveys?
- Stats from dev.tiki.org?
- Commit stats with Commit Tags?
- Feedback from the Triage Team?

1.4. Potential remedies to address root causes

Below is a list of potential areas to focus on. But we can only prioritize them once we have clear answers to the questions above.

If you want to fix more bugs, you need things like:

- More developers fixing bugs
- More testers

- A better process for bug prioritization
- Making bugs easier/faster to fix
- Catch regressions early, while they are easy to revert
- Reduce the introduction of bugs
- Etc.

1.4.1. Better triage

- Wishlist Team
- Ease Importance Priority
- Prioritisation of tasks (see Bernard's comment below)

1.4.2. Reduce the effort to reproduce bugs

- show.tiki.org Overview

1.4.3. Catch bugs early

- GlitchTip
- Pre-dogfood servers for Tiki 24 release process
  Pre-dogfood servers test have been very superficials (only testing if a Tiki page is editable, etc) years after years. A checklist of things to test (with checkbox for things completed) may help to perform more test and share the operation with several users.
- Test of "real life" Tikis using various configurations (features and settings)
  Tiki is filled with options and configurations parameters and each Tiki has its own life. Because of this, testing essentially our Pre-dogfood limit drastically the capabilities to catch early introduced regression or configuration conflicts between 2 features/settings that work well independantly. We need a panel of Tikis used in real life environment with different settings (that's the easy part per nature of our application) so we can have wider issues caught earlier and reported. There is a mutual interest here : Fixing productions Tiki that people want to maintain / Have more/better sources for reports. This also may encourage people to upgrade earlier.
  One clear drawback those are the contrary of Pre-dogfood servers where everything can be broken and accessed by our developers in a known environment.
- Encourage upgrading Tiki during the first stage of test
  Tiki community turned to be a conservative community and people are waiting (too long) before upgrading their Tikis. I have a few cases where Tiki "owners" waited several versions fearing breakage and then found themselves where they had to make a choice between "upgrading cost and pain on a not-popular Tiki" VS "new (better) site software migration". Tiki is likely to be ditched in such case because the "herb is always greener at your neighbour garden".
    - We could implement a "rewarding" system where selected Tiki site being upgraded and tested during beta and early release are "covered" by Tiki assistance for some duration. I can roughly imagine (TBD):
      - Selected Tikis with time limited access to our team (need better definition)
      - 15 days of assistance provided
      - Mobilised team to review, triage and fix (as possible)
    - A "Tiki tested" tag...

    - We need to be creative

1.4.4. Automation

- Continuous Integration
- Continuous upstream
- Continuous Testing Server
- Profiles tester (coming any day now)
    - Result: https://tikiwiki-ci.gitlab.io/tiki-profiles-tester/
    - Code: https://gitlab.com/tikiwiki-ci/tiki-profiles-tester

1.4.5. More developers

- Make it easier for new developers
    - Easy for Newbie
- Make it more interesting for developers
    - Better dev process
    - Professional opportunities
    - Learning opportunities

1.4.6. Clarify expectations

- Requirements
- Versions
- Tag some features as experimental or deprecated

1.5. Potential projects

1.5.1. Neglect

If the bugs are mostly in lesser used features and they are regressions caused by the lack of updates/maintenance. Ex.: A feature worked fine in Tiki9, but ambiant changes (upgrades to Smarty, PHP, Bootstrap, Composer, etc.) introduced an issue and it was not caught.

We have LTS versions so community users can remain longer on old versions. But it's not realistic to have community support much longer than current policy. And hosts will typically upgrade the PHP versions sooner or later.

These regressions can cause major issues. If it's adapting Tiki code to more recent PHP versions, it's usually easy to do. If a dependency is abandoned, it's usually more work/complexity.

The root cause is that these features are not important enough to the active community of developers.

Potential actions:

1. At each major version, do a smoke test of all features in Tiki with GlitchTip activated.

    - With minimum requirements? or also use as an opportunity to test maximum requirements?
2. Fix what is easy or Make a wish
    - When not easy/realistic to fix soon, tag that feature as "neglected" (or something)
    - Potentially nominate as an Endangered features

Action items that have been accepted and will happen

- Tiki Test Plan for new site admin