

Review of RFC-031: Module registry and construction context for wtf

What I verified

I read the RFC against `wtfutil/wtf` at `bb838c1` and checked every file:line reference it makes, plus the surrounding code the RFC doesn't cite: `app/widget_maker.go`, `app/widget_maker_test.go`, `app/wtf_app.go`, `app/module_validator.go`, `app/module_validator_test.go`, `app/scheduler_test.go`, `app/app_manager.go`, `help/help.go`, `wtf/wtfable.go`, `view/text_widget.go`, `view/base.go`, `modules/unknown/widget.go`, `generator/textwidget.go` and `.tpl` templates, and the `NewWidget` signature of all 86 `modules/*/widget.go` files (85 switch cases). I also checked for `init()` usage, build tags, and any documentation or CI step that inspects `widget_maker.go`.

The RFC's factual claims about the current code check out: the import block plus switch does span `app/widget_maker.go:9–379`; `MakeWidget/MakeWidgets` do span `98–399`; `buildDate/buildVersion` are exactly at `403–437`; `ModuleValidator.Validate` is exactly `27–41`; the unknown-widget message is exactly at `modules/unknown/widget.go:29`; and the specific claim that "six modules also store pages for their own dialogs" is exactly right — `airbrake`, `git`, `digitalocean`, `mercurial`, `todo`, and `weather` are the only widgets with a `pages *tview.Pages` struct field, out of 42 modules whose `NewWidget` accepts pages at all. The RFC is precise where it commits to specifics, which makes the gaps below more notable.

Findings

1. [High] `help/help.go` is not mentioned as a caller and its logic depends on exactly the contract this RFC removes. `help/help.go:26` calls `app.MakeWidget(nil, nil, moduleName, cfg, nil)` and then, at line 30, unconditionally calls `widget.CommonSettings()` and compares `widget.CommonSettings().Title == "Unknown"` to decide whether the `--module=X --help` flag was given a bad module name. This works today only because `MakeWidget` never returns `nil` for a bad-but-enabled module name — it falls into the `default:` case and returns the `unknown.Widget`. RFC §3.4 deletes the `unknown` package and changes `MakeWidget` to return `(nil, error)` for an unrecognized type. Section 3.5 says to update `app/widget_maker_test.go`; it says nothing about `help/help.go`. Under the new contract, `helpFor` will call `widget.CommonSettings()` on a `nil` widget for the exact case it exists to handle (an unrecognized `--module` name) and panic. This is a concrete, silent break in a real CLI feature, not a hypothetical one, and it needs a design answer (check the returned error instead of a magic string), not just a call-site update.

2. [High] Two more call sites use the current `MakeWidget` signature and will fail to compile. `app/scheduler_test.go:69` and `app/module_validator_test.go:79` both call `MakeWidget(nil, nil,`

`tt.moduleName, tt.config, make(chan bool))` with today's five-argument, single-return signature. RFC §3.5 only calls out `app/widget_maker_test.go`. These aren't hard to fix, but the RFC's "what this affects" list (front matter: `app/widget_maker.go, wtf/, modules/`) and its migration plan should say so explicitly — a contributor implementing this from the RFC alone would hit two compile failures the document doesn't warn about, on top of the `help` package break above, which is a design gap and not just a checklist gap.

3. [High] The config-hot-reload path turns a per-module typo into a full process exit, and the RFC never discusses it. `app/wtf_app.go:207-234`

(`watchForConfigChanges`) reacts to a write to the config file by calling `NewWtfApp` again while `wtf` is running. Today, if that edit introduces a bad `type: value`, `MakeWidget`'s default: case produces an inline unknown widget and the rest of the dashboard keeps running — the user sees one broken tile and 19 working ones. Under RFC §3.4, an unrecognized type becomes an error that `NewWtfApp` (per the RFC's own description, mirroring `ModuleValidator.Validate`) prints and then exits on. Since `watchForConfigChanges` calls `NewWtfApp` directly, that `os.Exit(1)` fires inside the live, running process the moment someone saves a config file with a typo — killing the whole dashboard instead of degrading one widget. This is a real behavior regression for a workflow the config-watcher exists specifically to support (live-editing `wtf.yml` while `wtf` runs), and neither §3.4, §4, nor §5 (Alternatives) mentions it. I'd want this decided explicitly: should config errors discovered via hot-reload be fatal, or should hot-reload keep the old widget set and just report the error without exiting?

4. [Medium] §3.4 states as decided what §6 lists as still open. §3.4 says flatly "`MakeWidgets` collects these errors. `NewWtfApp` prints them and exits," but §6's fourth open question asks "Should `MakeWidgets` return (`[]wtf.Wtfable`, `[]error`) so callers can handle partial failures?" These can't both be true — either the exit-on-any-error behavior is decided (in which case it belongs in §6 as closed, and finding 3 above needs an answer), or it isn't (in which case §3.4 shouldn't describe it as settled behavior). As written, a reader could easily implement §3.4 literally and consider the open question moot, without anyone having actually decided the question.

5. [Medium] The migration period's dual dispatch (registry + switch fallback) has no described precedence or merge rule. §4 says to "keep the switch as a fallback until all modules have been converted," but the `MakeWidget` shown in §3.4 is pure registry lookup with no switch at all — it's the end state, not the transitional state. With 85 switch cases and "roughly ten modules per PR" (so on the order of 8-9 PRs), the codebase spends a non-trivial amount of time with both mechanisms live. The RFC doesn't say whether the registry is checked before or after the switch, what happens if a module is accidentally left registered *and* still has a switch case (do they conflict, and does one silently shadow the other?), or whether that's even possible given how the PRs are supposed to be sequenced. This should be spelled out with the actual transitional `MakeWidget` body, not left implicit.

6. [Medium] RegisterModule has no duplicate-registration guard.

`registry[name] = factory` (§3.1) silently overwrites any existing entry. The

RFC's own motivation is partly "modules in other repositories could use the same registration mechanism," which raises exactly the scenario where a third-party module's chosen type name collides with a built-in's (or with another third-party module's). Right now that fails silently and non-deterministically (whichever `init()` runs last wins, and Go doesn't guarantee `init` order across packages with no dependency relationship in a way a contributor can predict). At minimum `RegisterModule` should panic (this happens at program start, before any user-facing state exists, so a hard failure is cheap and appropriate — this is exactly the pattern `database/sql`'s `Register` uses).

7. [Low] The "third-party modules" benefit in §1 is overstated. The RFC frames self-registration as enabling "modules in other repositories," and correctly rejects Go's `plugin` package in §5 for portability reasons. But without `plugin`, a module from another repository can only join the registry by being blank-imported into some `main` package built from source — i.e., a fork with an added import line, then a rebuild. That's not meaningfully different from today's requirement to add an import to `app/widget_maker.go` and rebuild; the registry removes a merge-conflict hotspot and a switch statement, which is real and valuable, but it doesn't deliver out-of-tree pluggability. Worth softening the claim so reviewers don't approve on a benefit the design doesn't actually provide.

8. [Low] Migration effort estimate likely undercounts touched files. At least 12 modules' own `widget_test.go` files call `NewWidget(...)` directly with the current argument list (e.g. `modules/bargraph/widget_test.go:29,117,285`, plus `azurelogs`, `cmdrunner`, `devto`, `feedreader`, `football`, `system`, `ping`, `mercurial`, `progress`, `stocks/yfinance`, `textfile`). Each of these needs updating in the same PR as its module's constructor signature change. This is implied by "update `NewWidget`" in §4 but not called out, and it's the kind of detail that changes what "roughly ten modules per PR" actually costs in review time.

9. [Low] This is the first `init()`-based registration in the codebase. A repo-wide search turns up zero existing `func init()` definitions outside generated/test code. That's not a reason to reject the pattern — it's idiomatic Go and well preceded elsewhere (`database/sql`, image format registries) — but it is a real, first-time trade-off for this project: "what modules exist and how they're built" moves from one linearly-readable file to an implicit graph of `modules/all.go` blank imports plus `N` `register.go` files, discoverable only by knowing the convention. Given the RFC's whole premise is contributor friction, this cost belongs in §5 next to the alternatives that were rejected, rather than going unmentioned.

10. [Nit] The §2 class diagram doesn't match `wtf.Wtfable`. The diagram shows `Wtfable` with `Refresh()`, `Name()`, `TextView()`. The actual interface (`wtf/wtfable.go:9–26`) has no `Refresh()` and composes `Enablable`, `Schedulable`, `Stoppable` plus eight other methods. Presumably intentional simplification for readability, but the RFC is otherwise scrupulous about exact

references, so an "abbreviated" note would avoid a reviewer flagging it as an error.

Design comments

The core idea — a name-to-factory map populated by `init()`, looked up by `MakeWidget`, with construction inputs collapsed into one `ModuleContext` struct — is a sound, minimal design and clearly beats the two alternatives it's compared against in §5 (code-gen adds a build step for a problem that's really about ownership of one file; a plain central map without `init()`-registration still leaves app importing every module). The `ModuleContext` struct is well-scoped: I checked every `NewWidget` signature in the tree, and `TViewApp/Pages/RedrawChan` is a complete and correct capture of what constructors use, with system's build metadata being the sole, correctly-identified exception. Keeping the registry in `wtf` rather than `app` is the right call and is genuinely cycle-free — I confirmed no module currently imports `app`, and `view` (which every widget already depends on) already imports `wtf`, so adding `wtf.RegisterModule/wtf.ModuleContext` calls to each module's `register.go` introduces no new import-cycle risk.

Where the design is thinner is at the boundary between "pure factory function" and the actual failure semantics of the running application. The RFC treats "return an error instead of building an unknown widget" as a small, mechanical change, but it's actually a behavior change with two live consumers (the CLI help path and the hot-reload path) that don't tolerate it as written, plus an unresolved question about whether startup failure should even be all-or-nothing (§6's own question). None of that is exotic to find — both `help/help.go` and `watchForConfigChanges` are one hop from `MakeWidget/NewWtfApp` — but the RFC's review of "current design" in §2 stops at `MakeWidget/MakeWidgets` and doesn't trace either downstream consumer, which is how both were missed.

Recommendation: request changes

The registry/`ModuleContext` mechanism itself is approvable as designed. I would not merge this RFC as-is because it asserts a stable end-state behavior (exit-on-unknown-type) without tracing it through two real call sites that break under it, and because two internal sections (§3.4 and §6) disagree about whether that behavior is even decided. Before this goes to implementation I'd require: (a) an explicit fix for `help/help.go`'s help-for-unknown-module path, replacing the "Unknown" sentinel check with the returned error; (b) `app/scheduler_test.go` and `app/module_validator_test.go` added to the affected-files list; (c) a decided, single answer to whether `MakeWidgets/NewWtfApp` fail fast or degrade per-widget, applied consistently to both the CLI-startup and config-hot-reload paths (the latter should almost certainly not exit the running process on a single bad module); (d) the transitional `MakeWidget` body for the mixed registry/switch period, with an explicit precedence rule; and (e) a duplicate-registration guard in `RegisterModule`. None of these require rethinking the core design — they're

the difference between a registry that replaces the switch cleanly and one that quietly changes what happens when a user makes a typo.