Engineering Maintainable WordPress Websites

Toru Miki

WordCamp Taiwan 2025

Not updated.
Cannot update.
Updates breaks custom codes.
10,000 errors per minute.
No one know why plugin x is used.
…etc

```php
// Main Unit Update Notification OFF
add_filter('pre_site_transient_update_core', '__return_zero');
remove_action('wp_version_check', 'wp_version_check');
remove_action('admin_init', ‘_maybe_update_core');

// Plugin Update Notification OFF
add_filter("pre_site_transient_update_plugins", "__return_null");

// Hide Unnecessary Menus
add_action('admin_menu', 'remove_menus');
function remove_menus()
{
  remove_menu_page(‘index.php’);
  . . .
}
```

Built to work, not to last.

Maintained to just run, not to improve.

Surely, maintainability matters. Let’s not create long term pains for both us and clients.

- Logs
- Principles of Plugin Selection
- Separation of Concerns
- Development Practices

Error Logs are Gold

Stop relying on guessworks!

Error Logs are Gold

- Logs tell you what actually happened – not what you think happened.
- Stop relying on guesswork!
- Make sure logs are accessible and viewable
- Make it part of your workflow

Principles of Plugin Selection

Look beyond the feature list to evaluate long-term viability

Principles of Plugin Selection

- Responsively maintained. Timely fixes (esp. security issues), and honest support. Not flashy but reliable.
- Prefer single-purpose. Opt for focused solutions. Avoid bloated.
- Secure, well-written (and scalable) code. Quality matters.
- Always keep in mind: You are effectively handing off functionality and maintenance of this feature completely to someone else!
  JA: 丸投げ　ZH-Hant: 丟包

Separation of Concerns

Separation of Concerns

Organize code so each part has a clear, single responsibility

Separation of Concerns

- Good Practices
  - Keep each function, class, or file focused on a single, well-defined purpose. (Single Responsibility Principle, SRP)
  - Theme = Presentation. Plugin = Functionality.
  - Custom features in dedicated, single-purpose plugin.
  - Functionality is visible and discoverable.

Separation of Concerns

- Bad Practices
  - Functions, classes and files handling multiple unrelated tasks.
  - Custom features buried inside functions.php
  - Mixing presentation and logic. E.g. custom post type defined in the theme
  - Hard to see what custom functionality exists or where it lives

Development Practices

Development Practices

Adopt tools and workflows from the border software engineering world

Development Practices

- Version Control
  - Use Git to track every change and maintain a clear, revertible history.
- Pull Request (PR) based workflow
  - Work through PR (even solo) to group related changes and trigger automations.

Development Practices

- Static Analysis
  - PHPCS
  - ESLint
  - Stylelint
- Testing
  - Unit test, visual regression test, etc.
  - Where possible. Prevent regressions, and ensure stability.

Development Practices

- Documentation
  - Write lightweight docs to ease future maintenance.
  - PHPDocs
- CI/CD
  - PR triggers running of the static analysis and tests.
  - Safer deployments – no need to manually update.
  - GitHub Actions.

Let’s raise the bar

– Software Engineering at Google