The IFD Versioning System

A Guide to Surgical Changes, Clean Checkpoints, and Confident Releases.

Three Version Types, Three Distinct Purposes

Minor Version (vN.N.X)

Purpose

Active Development & Rapid Iteration

Code Sharing

Yes (Link Back & Override)

Major Version (vN.X.0)

Purpose

Consolidation Checkpoint

Code Sharing

None (Self-Contained)

Release Version (vX.0.0)

Purpose

Production Deployment

Code Sharing

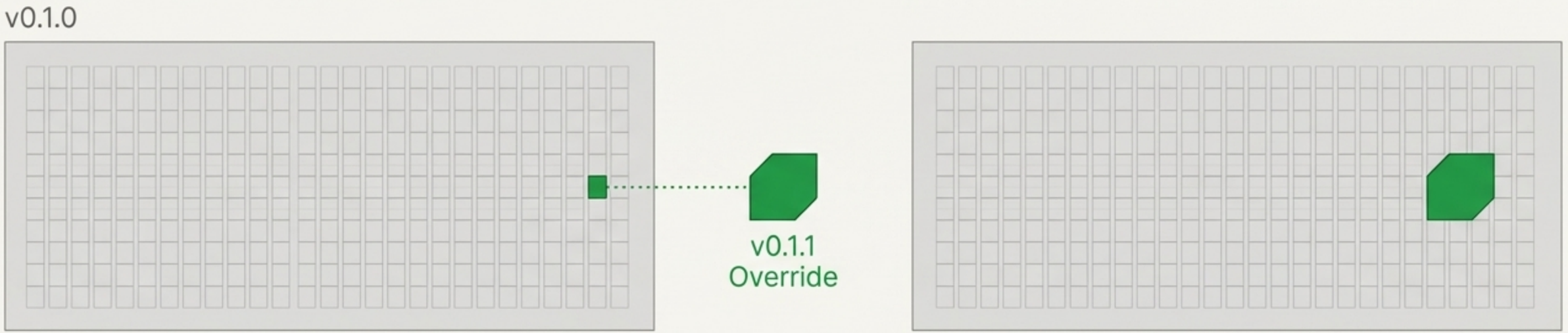
None (Exact Copy)

Each version type has specific rules governing its creation, content, and relationship to others.
Let's explore the path from a single change to a final product.

Minor Versions are Surgical Strikes for Active Development

"The unit of change is **the change itself**, not the file."

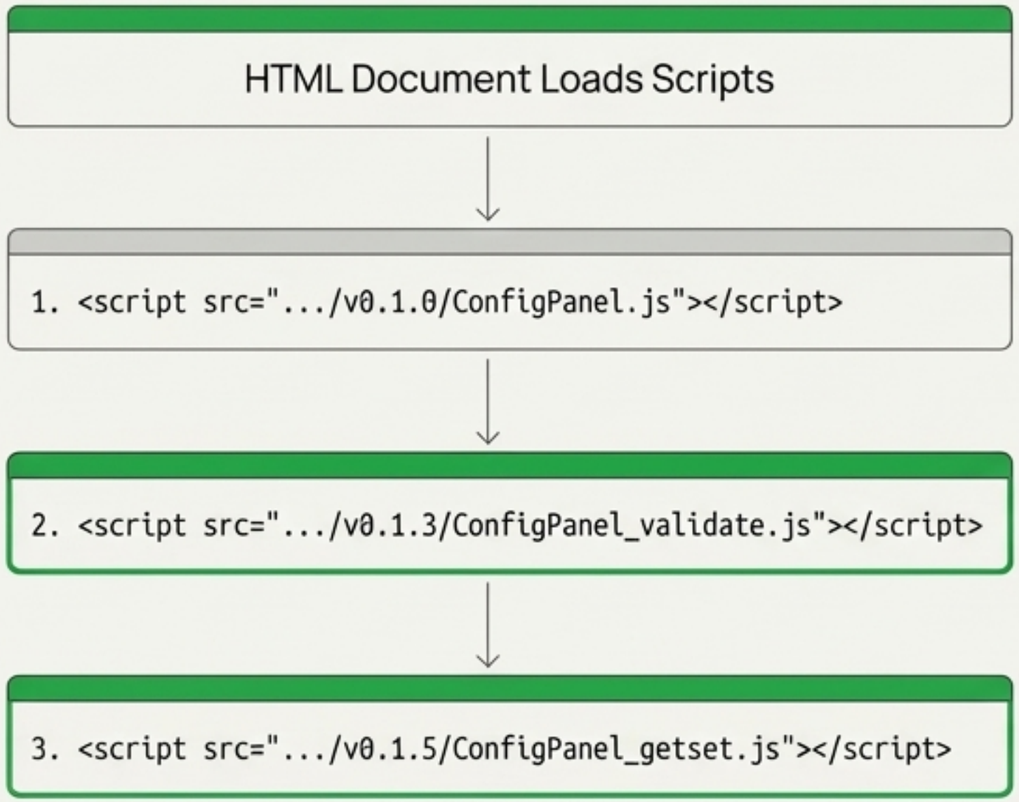
- **Purpose**: Where all active development happens. Each version adds a single, focused feature or fix.
- **Cadence**: Designed for rapid iteration, with cycles as short as 15-30 minutes.
- **State**: Potentially shippable at any point, as changes are additive and non-destructive.



How Surgical Overwrites Deliver Precision

The “Link Back & Override’ Rule in Action

Minor versions contain ***only*** the specific changes for that version. The base code is linked, and later definitions naturally win due to the loading order in the browser.



The complete, 20-method base class.

A file with *only* the `validateInput` method override.

A file with *only* the `getConfig` and `setConfig` method overrides.

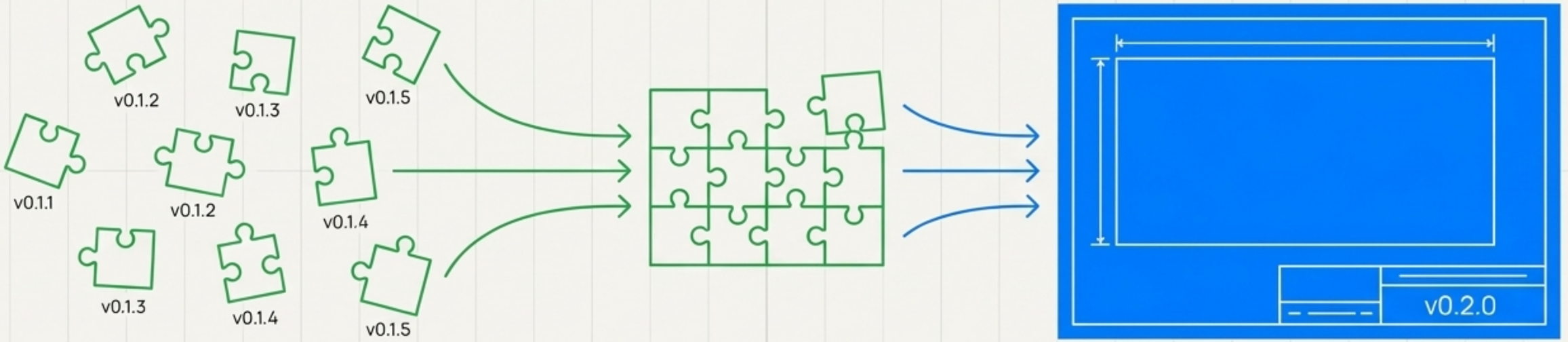
Result: The Final `ConfigPanel` Class Has:

- All 20 methods from v0.1.0.
- `validateInput` is replaced by v0.1.3's version.
- `getConfig` and `setConfig` are replaced by v0.1.5's versions.

Major Versions are the Blueprint Consolidation

Major versions are **completely self-contained.**

- **Purpose**: To merge all preceding surgical overrides into new, complete files, creating a clean and robust baseline.
- **Function**: Acts as a critical checkpoint for testing and review before a release candidate is considered.
- **Benefit**: Eliminates the complexity of a long chain of overrides, providing a fresh start for the next cycle of development.

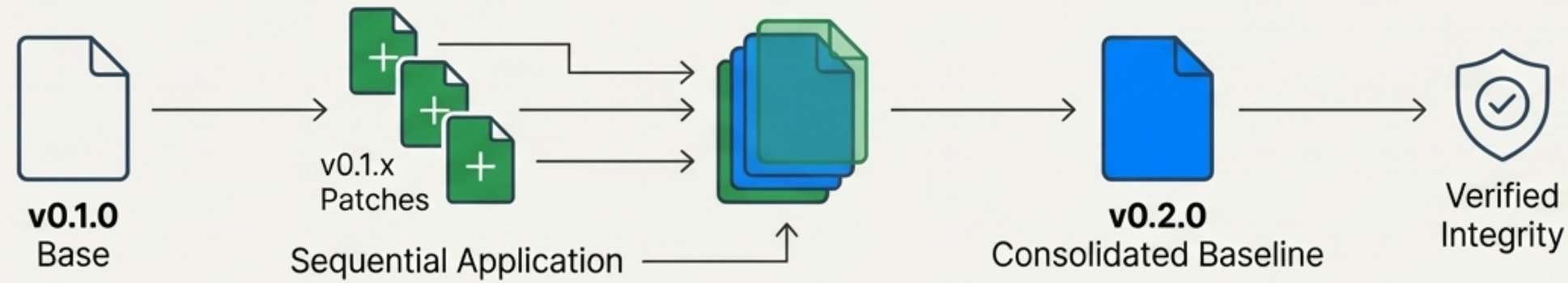


The Consolidation Process: Merging Patches into a New Baseline

From v0.1.x to v0.2.0

This is not simply copying files. It is an integration process where surgical patches are methodically applied to the base code.

1. **Start** with the base file from the previous Major version (e.g., v0.1.0).
2. **Apply** each surgical override in sequential version order.
3. **Integrate** the changes directly into the base file's content.
4. **Finalise** the new, complete file for the new Major version (v0.2.0).
5. **Update** all code imports to be relative within the new v0.2.0 folder.
6. **Verify** by running all tests to ensure full integrity.



Release Versions are The Final, Unchanging Seal

- **Purpose**: A simple, unambiguous marker for a production deployment.
- **Relationship**: An exact copy of the preceding Major version (e.g., v1.0.0 is a direct copy of v0.2.0).
- **Core Principle**: Guarantees that no last-minute changes, ‘quick fixes,’ or **unverified code** can sneak into the **production** build. It builds **ultimate confidence** in the release.

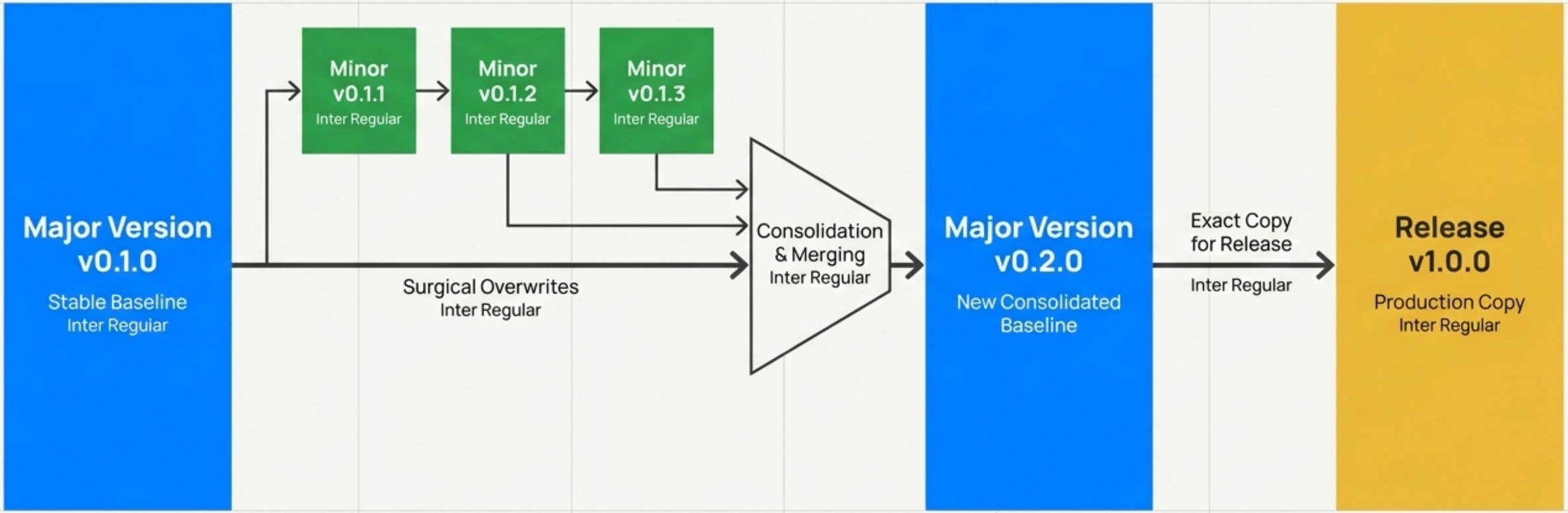
“What you tested IS what you ship.”

The Rule of Release: Only Version Numbers Change

WHAT CHANGES	WHAT MUST NOT CHANGE
✓ Version strings in code comments.	✗ No bug fixes.
✓ Version in `package.json` / config files.	✗ No feature additions.
✓ Version number displayed in the UI.	✗ No refactoring.
✓ Release notes and changelogs.	✗ No ‘quick improvements’.
✓ Documentation updates.	✗ Absolutely no ‘one more surgical fix’.

This discipline is the key to release confidence.

The Complete IFD Version Lifecycle

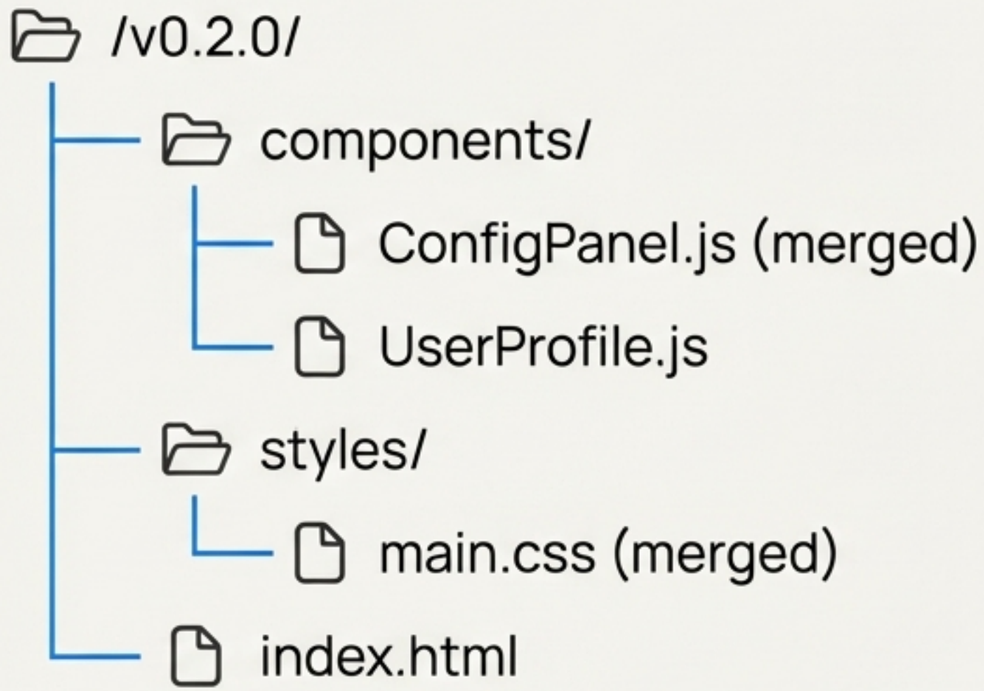


Folder Structures Reflect the Version's Purpose

Minor Version (v0.1.4) - Surgical & Sparse

Contains only the overrides. Relies on the base Major version for all other files.

Major Version (v0.2.0) - Complete & Merged



A self-contained, complete snapshot of the code with all changes integrated.

Design Your Code to be Override-Friendly

DO

Use prototype methods. They are easily and cleanly overridable.

```
MyClass.prototype.doWork = function() { ... };
```

Use CSS custom properties for theming. They are designed for overriding.

```
:root { --primary-color: blue; }
.header { color: var(--primary-color); }
```

Keep methods focused and small. Small targets are easier to hit.

DON'T

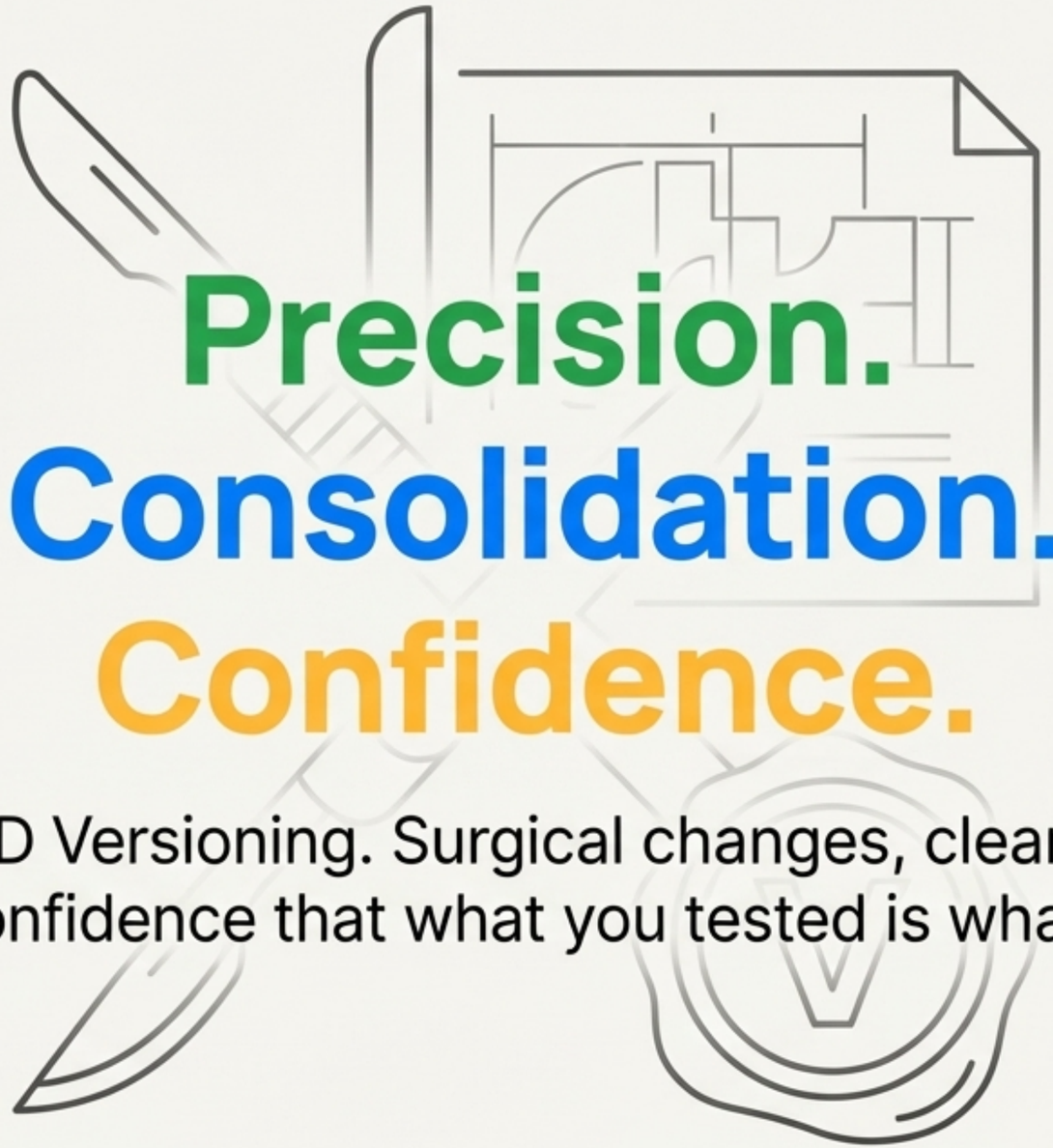
Use closures that hide methods. Private methods cannot be surgically replaced.

```
(function() {
  function privateMethod() {...}
})();
```

Inline everything into one giant method. This forces you to replace the entire block for a tiny change.

IFD Versioning at a Glance: A Quick Reference

Action	Version Type to Create	What Goes in the Version
Fix one broken method	**Minor** (e.g., v0.1.4)	Only that single method override.
Add a new feature	**Minor** (e.g., v0.1.5)	Only the new files plus any necessary overrides.
Tweak a CSS colour	**Minor** (e.g., v0.1.6)	Only the single CSS rule that changes.
Consolidate for testing	**Major** (e.g., v0.2.0)	All files, fully merged and complete.
Ship to production	**Release** (e.g., v1.0.0)	An exact copy, with only version strings strings updated.



Precision.
Consolidation.
Confidence.

This is IFD Versioning. Surgical changes, clean merges, and the confidence that what you tested is what you ship.