Overriding core limitations : how to handle it the right way

A bit of theory

When talking about managing an open-source technology integration, there are a ton of debates about “to do” or “no to do” adaptations to the code. Many of the exposed reasons often fall into a dogmatic discourse, the originating motivation is understandable : the lack of architectural mastering that set many of the moodle administrators and technical crew in a “black box” situation, i.e., having between hands an “opaque thing” we do not really know how it works. But if we respect the given rules, it should work.

All is about trouble of the technical crew assigned to making moodle work.

Directly opposed to this “prudent” way of managing the platform lifecycle, is usage and requirement for doing “better”, “clearer”, more “user friendly” user experience. Often this view stucks on core limitations, i.e., unforecasted consequences of code design, unexpected or not imagined uses cases, etc. Moodle has a LOT of tuning and overriding possibilities and techniques, thus those cases of hanging on a “really tricky override case” are few.

The purpose of this chapter is to demonstrate that locking on a Moodle Intrinsic limitation is not a pure fatality. All depends on a fair evaluation of the balance between Maintenance Trouble amount vs. Usage Benefice. Most of the reluctance factors to accept core overrides are measured by the “factor of Trouble occasionned for Moodle tech mainteners crew'.

We propose further some theorical approach of the concern:

First will we propose a theorical definition of the “Technical Impact” of a design decision. The **technical impact** (Imp) roughly determines how amount of effort will be necessary to maintain properly a working application:

$$Imp = \frac{Nop * Xop * Of}{Aut}$$

The impact unit should be not far from some Energy measurement.

It can be modelized roughly as a function of increasing factors: Number of maintenance operations - Nop, complexitxy of operations - Xop, and probably an organisational factor that increases when both number and complexity grows. Organisational factor can include f.e. an increasing of the number of actors in the operations and the loss in coordination.

Opositely, there is one big factor that helps reducing the technical impact : Automation (Aut). The more automated you are (including also power tools, not necessary fully automated), the less the technical impact will be a trouble.

Now we can address the notion of “Trouble”. “Trouble” is a subjective notion, and wil not affect all actors the same way. Maintainers will have “Trouble” in having to maintain complex things and make it available for users. Users will be troubled in not having friendly enough or unefficiant tools,

stakeholders will be troubled to get informed and harassed that thing do NOT WORK well.

Customcripts

Customscripts are a “soft” way to get over the moodle core limitations. Customscripting is a standard mechanism to get core code replaced by a customized version, WITHOUT interfering with core code from the official Git or distribution.

By defining a directory (often `/customscripts`) at moodle code's root (or outside) and adding a key:

```
$CFG->customscripts = '<moodlepath>/customscripts';
```

In your config file, then any request to a php script say: `https:<mymoodle>/a/b/c.php that encounters an existing file names '/customscripts/a/b/c.php' will be derouted at end of the setup.php'' initialisation to that file, includes the file, then continue normal processing. ====Core patchs==== ====Tracking patches and customscriptings needs in plugin design==== Back to general index`

From:
https://docsen.activeprolearn.com/ - **Documentation Moodle ActiveProLearn**

Permanent link:
https://docsen.activeprolearn.com/doku.php?id=start:overrides&rev=1770578839

Last update: **2026/02/08 20:27**