# XBlock Lessons Learned

This document is an overview of lessons that I took away from XBlocks and some concrete examples for how those lessons might translate into a new system. This is not a comprehensive proposal and many ideas are incomplete. This is not based on recent, rigorous, and unbiased testing and surveys. This is not edX's formal position on anything. This is not a technical roadmap. This is me trying to summarize years of arch discussions, personal frustrations, and interactions with third party XBlock developers. These are my recollections and opinions. I hope it's a useful starting point for conversation.

For those that want to skim, I've put takeaway summaries in 💡 bullet points for many longer sections.

# Unrealized Goals

XBlock was never completely developed into its original vision, and a number of compromise solutions have solidified over the years. That original vision isn't necessarily *correct*, but knowing it may help as we're thinking about where we go from here.

## Semantic Naming

What we call a "vertical" should have been "unit", rather than indicating layout on a page. This was the original intent, but somehow got lost or miscommunicated in the rush to build Studio.

## Extremely Dynamic Behavior

- 💡 XBlocks were built with a "query the smart object" philosophy intended to promote highly dynamic, adaptive content.
- 💡 We had to dial this back to something much more static in order to maintain performance, stability, and a simpler interface for analytics and mobile.
- 💡 We've shifted from "ask the XBlock anything" to "have XBlocks push data into and query specialized services".
- 💡 As part of that shift, sequence-like responsibilities should eventually be moved out of XBlock.

I have an article on the problems we encountered with maintaining the XBlock grading system where I go into much deeper detail, but the short of it is that you can't make any performance or stability guarantees when you have to query dozens of plugins at runtime to serve simple requests. To address these problems, we gradually flipped the data relationship and made XBlocks push data into dedicated systems with their own data models (e.g. grades, completion), rather than querying the XBlocks at runtime.

This was one of the most painful concessions because it severely limited the intended power of the framework. A/B tests were part of the original prototype rollout, but did not appear again in a usable form for years.

As part of this architectural shift from direct XBlock queries to specialized services, we're planning to shift sequencing responsibilities from XBlock and into new systems that will allow for more dynamic capabilities while still being scalable and performant.

## More Granular Composition

- 💡 People creating custom content types should not have to reinvent the UX and behavior for common things like score display and "show answer" logic.
- 💡 It should be straightforward to add a new feature (e.g. hints, inline discussions) that apply to a wide group of content types in a pluggable way.
- 💡 Capa allows for multiple responses to be grouped into a single submission (i.e. a multi-part problem). Other XBlocks can't participate in this yet.

XBlock was not only intended as a replacement for XModule, but also for CapaModule, subsuming its concept of input and response types into a unified component model that would stretch from containers (courses, sequences) down to individual buttons. That would mean that theoretically, all graded types might use the same "SubmitBlock" that provides the UX for the submit button, logic for number of attempts and score display, etc. An example of this sort of thinking is available in xblock-sdk's problem module. There is a pseudo-working version of problem XBlocks with children in Problem Builder, though it has to hack around the lack of real support for this usage pattern in edx-platform.

We never even completed the original XModule to XBlock migration, and so we never got to the point of implementing this functionality. This led to a lot of ad hoc replication of code and inconsistent behavior around handling due dates, displaying points earned, hinting, and showing answers. Rather than having an enumerated set of states that the problem can be in (e.g. past due, answers available), XBlocks inconsistently reimplement this logic based on inspecting various attributes.

That being said, adding this functionality may have also added even more complexity and performance headaches to the XBlock runtimes.

## Separation of Policy from Content

- 💡 Clearly separate content (e.g. the text of a problem) from policy (e.g. its due date).
- 💡 That policy should be modeled with services and not simple read/write data fields.

In order to facilitate reuse, we wanted to separate the content that didn't change between course runs from the policy settings that would. So for instance, the text of a problem usually remains the same between courses, but the due date constantly changes. This was the purpose of having the "content" vs. "settings" scope for XBlock field data.

The first thing that went wrong with this approach was that there was no enforcement, so the settings scope was used in many places that should have been content scoped. For instance, just about everything about a VideoModule is settings scoped, including transcripts and the locations of the videos themselves. Some of this may have come from copy-pasting, some from being simpler to set up Studio editing, or possibly some obscure bug in how we handled content scoped content in the early days. Regardless, the situation today is such that the settings scope is used far more often than it was ever intended and usage is so muddled that we can't really infer anything meaningful from whether a field is scoped to content vs. settings.

A less obvious issue that came up was that the life cycle and usage of course policy settings vs. content are very different. I have a much longer blog post about XBlock Field Data Complexity that covers this in more detail, but the short version is:

- Content authors and course teams overlap but are not necessarily the same.
    - CCX has course teams that cannot author content in Studio at all, but will modify things like course schedule and grading policy.
- Content is very key-value oriented. There is a problem and it has some text. It might be summarized or indexed for searching, but the query patterns are fairly limited -- write to it in Studio and read from it somewhere in the LMS when you need to display it to the student.
- Course policies are much more sophisticated and are both read from and written to by the LMS. When is that assignment due for that student? Did they get an individual due date extension? What are all the assignments that are due this month for that student? What are all the due date extensions that this staff member has authorized for this assignment?
    - Things like "due date" and "number of attempts allowed" are windows into bigger systems that have their own logic. Modeling them as simple key/value pairs made it much harder to create these basic sorts of queries.

In order to walk back some of these decisions in a backwards compatible way, we're using the pattern of keeping Studio data the same but translating that data into a more easily queryable form when course content gets published to the LMS. This is the approach that edx-when takes.

## Separation of Run from Course

💡 The LMS and Studio were meant to understand Course vs. CourseRun, but in practice they only understand CourseRun today.

The original prototype course on what became Open edX had two concurrent runs that supported an on-campus MIT course and a MOOC. This was in the XML-only days, so it was done in a hacky way using symlinks and Mako templates, but that conceptual separation between a Course and the Runs of that course existed from the earliest time.

Unfortunately, this concept carried over to Studio in a very incomplete way (probably because Studio's development schedule was ridiculously crunched). Studio was originally meant to edit

Courses, not Course Runs, and if you look at its treatment of Old Mongo courses, you'll see that the URLs don't have any runs in them. But in practice, there was no good separation of Course and Run within Studio and trying to build that separation was adding complexity. It quickly became a place to edit CourseRuns only, with ugly glue code to derive CourseRuns from Course identifiers. By the time SplitMongo came around, it was firmly a CourseRun-only effort, even if they were still called "Courses".

## Omni-Framework

- 💡 Should everything in courseware (tabs, courseware, profile pages, etc.) be an XBlock? (No, absolutely not.)
- 💡 Some features want multiple UIs (e.g. inline discussion and discussion tab).

One of the early views of XBlock was that we should have a unified extension mechanism that would be at the core of basically everything in Open edX—your tabs would be XBlocks, analytics extensions would be XBlocks, etc. I'm strongly opposed to this in one sense (that the XBlock class and runtime are everywhere) because it leads to making a big, non-performant system that you can't make any guarantees about. On the other hand, it does make sense to have a feature that uses multiple extension points at the same time, and to have some coherent way to group changes together. We currently do this via Python entry points, and would want some equivalent for any future plugin framework.

## Self-contained Editing

- 💡 XBlocks were meant to be completely self contained objects that could edit themselves, which was only partially achieved.
- 💡 It's less clear to me today whether we want this sort of coupling even in leaf components, particularly in a potentially much more granular world.
- 💡 Decoupled editing would be helped by stable formats and policy separation.

XBlocks were meant to be self-contained in terms of editing. They would know how to serialize themselves. We're in a very mixed state with a lot of stuff hardcoded into Studio today. Structural elements are entirely edited via custom code and not a part of CourseModule or SequenceModule.

While the batteries-included approach of this is nice (particularly if the format is rapidly evolving), it limits the ability to plug in third party editors and can weigh down the core modules with a lot of extra code.

# Unexpected Issues

These are either things that we didn't think about, or that turned out to be much more complicated than we initially thought they'd be when we punted on them in favor of more urgent concerns.

# OLX Format Compatibility

- 💡 Treat file formats as first class products and not serialization details.
- 💡 Store content in its official file format somewhere in the database, even if we end up having a separate optimized representation.
- 💡 Have a clear vision for versioning, even if it's not immediately implemented.
- 💡 Separate the definition of the standardized container format, the core supported types, and any third party types.

I have always been frustrated when conversations about standardizing OLX are framed with the intention of creating an official spec along the lines of what IMS does for LTI. On the surface, it's a perfectly reasonable thing to want. OLX is our import/export mechanism and many people have written scripts that rely on the stability of the file format. It absolutely should be better documented and tested than it is today. The problem is that OLX is an ad hoc serialization format that was tacked onto as new requirements arose, and it conflates a few different concepts:

1. A container format describing the structure and policy of a course.
2. Specific serialization for well supported, "core" content types.
3. Generic serialization for unsupported types.
4. Custom serialization for unsupported types.

There are many aspects about running a course (start date, policies, etc.) that are simply crammed into the CourseModuleDescriptor's XML attributes because we had no other place to store them at the time. We have JSON-in-XML attributes, sometimes doubly-escaped. XBlocks can have child elements but they can also have any arbitrarily complex XML for a serialization format, and it's impossible to tell whether a blob of XML is describing one complex XBlock or an XBlock and a dozen children unless you're actually running the XBlock runtime. New fields for individual types are added all the time without versioning. Compatibility is often unintentionally broken (despite our best efforts) because our ability to detect edge case regressions is almost non-existent.

When it comes right down to it, XBlocks think in terms of Field Data and only incidentally serialize to and from XML. This is an important distinction because we don't actually store the OLX in any permanent form in our databases (except for the subset that is Capa). When you import OLX, it's parsed into a JSON-like structure and stored in MongoDB. If you later remove the XBlock, we have no way to export that OLX back out in any meaningful way. When we accidentally break compatibility, we often can't even export out the content we have in the database to see what its state is. The report we get is "this course won't display or export" and the only recourse is to examine MongoDB.

## Strict Leaf vs. Container Hierarchies are Limiting

- 💡 We should be able to nest elements in semantically meaningful ways (e.g. a video in a problem), instead of strict container-primitive hierarchies.
- 💡 This is one of my more controversial opinions. Another view is that using simpler grouping wrappers within Units can accomplish this more simply. The lines between reusable pieces and their containing context are very blurry in different situations.
- 💡 I think a lot of this depends on whether it's natural in whatever implementation path is chosen. It's not really for XBlocks today.

The XBlock framework assumes that you're either a self-contained leaf node that renders HTML, or you're a container that calls the rendering method on your child XBlocks. A Unit displays all its children in a flat listing. It's a simple model, and it makes it easier to make editors. But sometimes the blocks would be more semantically meaningful if they didn't have to obey that strict hierarchy. For instance, take something like this:

```
<vertical>
    <html>Now that you've done XYZ, try analyzing the following video:</html>
    <video>...</video>
    <problem>What techniques are demonstrated by this video?</problem>
</vertical>
```

That problem is explicitly about the video, and it's useless without it. You could manually embed the video inside the problem itself with HTML, but then you'd be getting the raw HTML version of it, and not the XBlock (with its various UI and analytics enhancements). It would be nice to be able to do things like:

```
<unit>
    <p>Now that you've done XYZ, try analyzing the following video:</p>
    <problem>
        <video>...</video>
        <prompt>What techniques are demonstrated by this video?</prompt>
        ...
    </problem>
</unit>
```

A lot of the time when we talk about the dichotomy of sharing something on its own vs in the context of a Unit, it's a byproduct of the granular piece not having sufficient context to stand alone, but the containing Unit having too much context that is specific to its usage in a course. (Though that's a crude simplification -- sometimes the problem itself makes no sense without other material in the Unit or even containing Sequence.)

In a similar fashion, it should be possible to place inputs in a table (say you have a half filled in table of values and you expect them to fill in the rest)—mixing what you'd think of as HTML markup with the input/output of the problem.

## Content Changes and Student State

- 💡 The new system should guide developers to think about and handle situations that arise from content changing over time as course teams fix mistakes.

Developers creating XBlocks usually operate under the assumption that content will remain static after it has been created, and this is often wrong. What happens when a user's stored answer to a question is an option that no longer exists as one of the possible choices? What happens to a peer graded assessment that was made using a rubric that was later updated?

Sometimes the solution has been to simply catch and reset student state when it describes an impossible operation on the updated content. Sometimes we've replicated content state in the database as a stable snapshot. For instance in ORA2, the grading rubric for any given peer assessment is captured at the time the assessment was made, so that an updated rubric from the content will not affect the state of previously created assessments.

There is also another concept here around major and minor changes to a problem, an idea that exists in other LMS products we've seen. A minor change to a problem does not affect the validity of student state captured against it. An example of this might be a minor typo. A major change does affect the validity of student state. Such a thing might be catchable in an automated way (e.g. the answer options were changed), but some of it might require manual intervention to catch (e.g. forgetting to add "NOT" in the question).

## Unique Addressing (Trees vs. DAGs) and Shared State

- 💡 Course teams want intelligible content identifiers and URLs, not hashes and UUIDs.
- 💡 For user state storage, we want a stable ID that won't change if it gets moved around in the course.
- 💡 Decoupling browser addressability and state storage would give us more flexibility (both use the usage key today).
- 💡 There are use cases for both user state sharing and state isolation for the same content across different contexts, but we should explicitly model these without giant DAGs.

Courses are usually described as trees, but they're actually directed acyclic graphs. One motivating use case for this is that you can have two usages of the same problem sharing the same user state. So for instance, you can have a circuit that you gradually build over time in different sequences over the course.

However, this is not supported in Studio today and the way this is done with XML imports gives a great many headaches. What policies (start date, due date, etc.) are inherited by a problem that's in three different sequences? If we're given a request to "go to problem XYZ", which of the three instances are we going to point to? If we allow policy to inherit differently based on which instance of the problem we're seeing, what happens if the user state stored against one instance is incompatible with the policy in another?

There is also an argument for having isolation of student state even for the same problem, in case you want to check that they still remember something. In either case, having an explicit separation of what this problem wants to save across contexts vs. what is context specific would be helpful.

## Student Collaboration

- 💡 XBlock doesn't have good facilities for allowing multiple students to collaborate on work to be submitted.

Course teams want to be able to assign group projects. They want to allow students to chat with each other and assess one another. Doing this in XBlock can be pretty hacky. One way used to share data between students is to make a user_state_summary scoped field that's a pointer to a file system and use different directories in that file system to represent private user mailboxes. Another tactic has been to create actual database tables and an XBlock that understands those relationships (like ORA2). Some XBlocks have explicitly used the StudentModule interface to do cross-student state queries, which opens us up to security concerns.

## XBlock Class Does Too Much

- 💡 Avoid having one all-encompassing base class to extend.
- 💡 A Hints Button and a Problem type are not the same thing, even if they share some low level functionality.

When you create a subclass of XBlock, you've got one space for all your model state fields and view handlers. That was intended to make things easy to get started for new developers. Unfortunately, larger XBlocks like ORA2 end up being spread across a dozen mixins that get combined into a giant XBlock class with a hundred attributes and methods on it, making it really difficult to work with.

## Integration Points Around Content

- 💡 Pluggability means not just novel content but allowing introspection/decoration around content.

Features frequently want to wrap around or annotate content in some way. This can be at the leaf level (notes, problem analytics), the Unit level (bookmarks, discussion), the sequence (proctor messaging). Partners would also like to be able to make their own annotations to the data (e.g. their own identifiers for content) and have that flow through to analytics. Some of this is currently implemented with XBlock Asides, while others are baked into the UIs for Unit and Sequence.

## Migration Pain

💡 The XModule + XBlock compatibility work greatly complicated the system.

For years, edx-platform has run in a mixed mode of XModules and XBlocks, leading to some very confusing code to make the two systems run together (search the wiki for “peak confusion”). We are finally making headway to finish that conversion due to OpenCraft’s Blockstore-related contributions, but it’s a cautionary tale. The XBlock runtime concepts themselves are not that complicated, but making them work together with XModules led to a system that was so difficult to work with that most people avoided it altogether.

## Sequences are too Generic

💡 Assignments should be a first class concept and deserve their own API.

An ungraded sequence of videos and a homework assignment are different in some pretty fundamental ways. Having that factored out into different classes is useful. A lot of information is inherited down through sequences and into leaf XBlocks (like start and due dates), and this was a fast way to get a generic “write it once have it available in many places” behavior. But it means that we never really had to iron out a real API between say an assignment and individual problems within that assignment.

## Source Code and XBlock Fields Translations

[Omar Al-Ithawi added:] Source code translation in XBlocks is not a first-class feature. It’s often supplied as an afterthought and most of the XBlocks today (2019) don’t support internationalization.

XBlock fields translation is not possible for now. It is being discussed by a couple of community members but it’s not an active topic by any means.

## Content Translations

Some folks want to run a course in multiple languages concurrently. How to do that at the content level?

## Naming Confusion

Let’s call any new system an all caps acronym or a word with just one starting capital letter in it. There are at least half a dozen variations of “XBlock” and “Open edX” floating around (xBlock, XBlock, xblock, Open edX, OpenedX, openedX, etc.). 😛