COURSEUP: HUMAN READABLE COURSE LANGUAGE

Micah Taylor

Kixor Technologies

micah@kixortech.com

ABSTRACT

We present a new language, `CourseUp`, which is designed to allow easy defining, reuse, analysis, and sharing of course materials. Our language is based on proven web authoring technology, allowing materials to be readable and writable by both human and machine. In this paper we present a brief description of the language, show examples from undergraduate Computer Science courses using `CourseUp`, and describe the impact of using the language.

INTRODUCTION

Courses are often authored and presented in Learning Management Systems (LMS) like Blackboard and Moodle. These tools are very useful for allowing instructors to develop materials and for presenting course materials to students. The materials for LMS courses are stored in instance-specific backend databases. At the same time that the first LMS tools were being created, programming tools were becoming much more sophisticated. Modern code storage and sharing communities were growing alongside web technology and development languages. These coding developments gave birth to patterns which are well established today: easy tracking and forking with tools like git, rapid sharing and collaboration in communities like Github, and fast, convenient authoring with domain specific languages like Markdown.

We wish to leverage modern programming tools and practices to improve the course development process. We propose a new domain-specific language for defining courses and materials.

PREVIOUS WORK

Learning Management systems (LMS) are widely used, with moderate improvement in student experiences [**?**]. LMSs require significant institute investment for initial use [**?**] and materials can be difficult to transfer between systems. For example, once a course component is designed in one LMS, it is very difficult to migrate that component to another presentation system [**?**][**?**].

Storing and transferring LMS course materials has been investigated by ADL, resulting in several related specifications [**?**]. Unfortunately, all are API focused and not targeted as human readable design languages. Previous XML standards have been designed for course description [**?**], but have failed to gain traction in the community. For example, SAIL [**?**] is a language for describing course materials and, while human readable, required significant use of LaTeX. This required that a course designer already know the complex LaTeX typesetting syntax.

Still, domain specific languages that carefully target a specific audience have had great success in many areas [**?**]. For example, the MATLAB language, LaTeX , and RenderMan have been used for decades in matrix math, typesetting, and graphics respectively. Recently, more human focused markup languages like Markdown [**?**] have become widely used and adapted to specific domains, like R Markdown for statistics and

Bookdown for book publishing.

These types of languages can also leverage modern development tools and code communities. The availability of robust distributed software repositories have led to a culture of rapid iteration and sharing in the development community [**?**]. These tools also allow precise and detailed change tracking and analysis. Coupled with test based design, such tools can automatically find code changes that contribute to an error or issue [**?**]. A domain specific language for course design would leverage these advances. With the goal of reducing load on designers and allowing them to focus on course content, we propose a language with four design goals: human readability, location flexibility, material comparability, and automation.

- **Human readability** ensures that documents can be easily used by most people without requiring complex editing tools.
- **Location flexibility** allows designers to share and reuse content without needing supporting tools or personnel.
- **Material comparability** enables changes to be tracked and measured by source repositories and the communities that use them.
- **Automation** requires that rote actions (i.e. date management) be handled by the interpreter and not the designer.

LANGUAGE OVERVIEW AND EXAMPLES

`CourseUp`'s design goals inform the language specification. In this section, we present some samples of the specification and give examples their use. The primary components of `CourseUp` are:

- A **course hierarchy** defined by the file structure
- A **global configuration** for the entire course
- A **course specific language** for defining course materials

Course hierarchy

The hierarchy for `CourseUp` courses is derived from the arrangement on the local filesystem. This allows the user to easily modify and move materials. An example structure is shown in Figure 1.

```
Homework1\content.md
         \matrix.png
Homework2\content.md
         \example.js
         \sine_wav.mp3
Quiz1\content.md
Quiz2\content.md
config.yaml
schedule.md
```

Figure 1: CourseUp hierarchy

```
CourseTitle: "Sample CourseUp Course"
Resources:
  - ResourceName: Schedule
    ResourceURL: /
  - ResourceName: Syllabus
    ResourceURL: /syllabus/
FirstCourseDay: 2016-08-10
LastCourseDay: 2016-12-04
Breaks:
  - LastBeforeBreak: 2016-10-12
    FirstAfterBreak: 2016-10-15
```

Figure 2: Global config data

Global configuration

We wish to minimize setup and reuse costs, so we use the YAML format for storing course settings. YAML is designed for human use, while still being machine parsable. Some of the `CourseUp` basic configuration commands are presented below. An example using these settings is shown in Figure 2.

CourseTitle: A string that defines the course title.

Resources: A configuration section that marks the start of course resources. These resources are course materials that are potentially useful from any course section.

ResourceName: A string that specifies a resource name.

ResourceURL: A string that specifies a resource URL.

FirstCourseDay: A date in the format yyyy-mm-dd that specifies when the course starts.

LastCourseDay: A date that specifies when the course ends.

Breaks: A configuration section that marks the start of the break options.

LastBeforeBreak: A date that specifies when a break begins.

FirstAfterBreak: A date that specifies when a break ends.

Domain specific course language

Extensions to Markdown form the main description language for course materials. Figures 3 and 4 show example quiz code and output, while Figures 5 and 6 show example lesson code and output. Some of the Markdown extension details are presented below.

LaTeX: Only LaTeX math mode is fully supported. Inline LaTeX math statements are delimited between `\(` and `\)`. Math mode statements are delimited between `$$` and `$$`.

Rubric: Rubrics are specified as Markdown tables with the table title set as `Rubric`. The first column represents the criteria; each other cell contains the quality level description and the level score, separated by a colon.

Data entry: Space for data entry can be specified with `\inputBox{x,y}`, where x and y are measurements in units such as `em` or `mm`.

Solutions: Similar to LaTeX conditionals, solutions can be defined between conditional tags using `\ifsolution`, `\else`, and `\fi`. The conditional is fulfilled based on solution dates set in the calendar. This allows solutions to be automatically shown after the assignment is due.

Pagination: Page breaks in printed material can be added with the command `\pagebreak`.

```
\nameEntry
\mailboxEntry
\dateEntry

Quiz3
---
1.  What OpenGL version does your
    computer support?
    \inputbox{40em, 2em}
2.  Create a list of vertices that
    could bound a 3D rectangle with
     area of 4 that has a corner at the
    origin and lies in the X plane.
    \inputbox{40em, 4em}
```

Figure 3: CourseUp quiz code

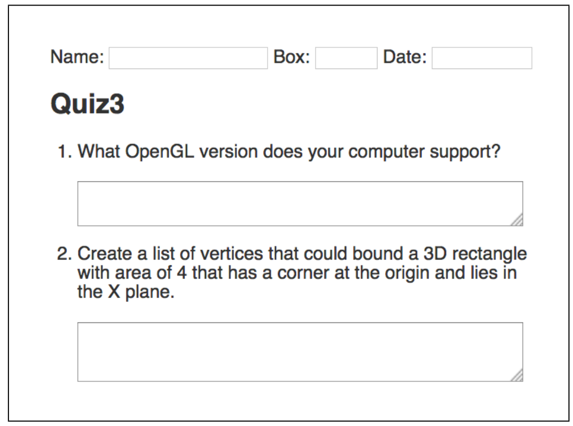

Figure 4: CourseUp quiz output

```
Camera
---
If you make sure the look and up vector
are normalized before building the
basis, then the resulting camera
basis \( \mathbf{C} \) is orthonormal:
$$
\mathbf{C} =
\begin{bmatrix}
u_x & v_x & w_x \\
u_y & v_y & w_y \\
u_z & v_z & w_z \\
\end{bmatrix}
$$
```

Figure 5: CourseUp lesson code

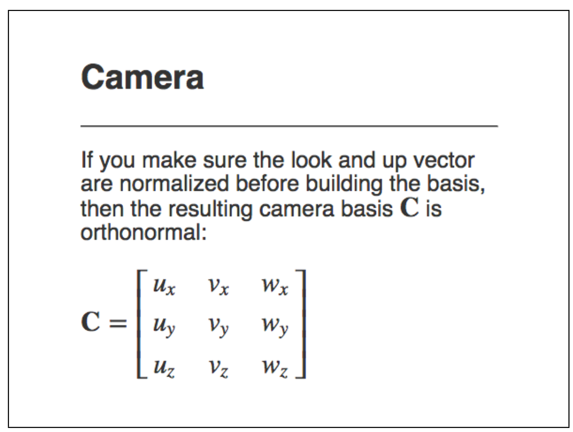

Figure 6: CourseUp lesson output

`CourseUp` can build flexible calendar schedules. Calendars are defined by the `\calendar` command. If a document has this command, `CourseUp` checks for special calendar commands. The calendar commands specify course session dates, due dates, and solution reveal dates. Based on the settings in the global configuration, all dates can be generated automatically. Figures 7 and 8 show an example schedule definition and output, respectively. The calendar commands are specified as follows.

Session: Emits the session date, according to the configuration file, then advances the internal date counter to the next session date.

+*string*: Specifies a reference to material in the course hierarchy with name *string*.

+due *integer*: Specifies that the current item is due in *integer* days.

+sol *integer*: Specifies that the current item solution is revealed *integer* days after the due date. The resulting date is used when evaluating conditionals like `\ifsolution`.

```
Session:
* [MIPS green sheet](pdf/Green_Card.pdf)
* Intro to MIPS assembly
 * Read 2.1--2.3
 * Practice 2.1--2.6; 2.9--2.10
* Representing instructions
 * Read 2.4--2.5
 * Practice 2.12; 2.14--2.18
* +HW2 due +2 sol +2
```

Figure 7: CourseUp schedule code

2: Tue Nov 28

- MIPS green sheet
- Intro to MIPS assembly
  - Read 2.1–2.3
  - Practice 2.1–2.6; 2.9–2.10
- Representing instructions
  - Read 2.4–2.5
  - Practice 2.12; 2.14–2.18
- HW2 (due Thu Nov 30)

Figure 8: CourseUp schedule output

RESULTS

`CourseUp` has been tested in 20 sections of three courses at an engineering university: *Computer Architecture*, *Computer Graphics I*, and *Computer Graphics II*. In this section, we highlight some examples of the unique aspects of using `CourseUp`.

Tracking changes

The final project in the university's *Computer Architecture* course is to build a processor and the project is quite challenging for students. Several offerings ago, students evaluations noted:

"Please make the Milestones clearer"

"The milestones are difficult to understand, what exactly is wanted for deliverable"

```
----------- milestone2/content.md -----------
@@ -20,12 +20,14 @@

Next, list and describe the components [-required-]{+that will be needed+} to
implement your RTL.

[-Make sure that resources are not over used.-] {+Do not describe how the parts
are connected; you just need build a 'shopping list' of generic parts.+}
```

Figure 9: A `git` repository diff of milestone revisions

Since `CourseUp` files are text based and the course is stored in a code repository, it was simple to identify and track changes as the milestones were revised (Figure 9). This allowed the instructor to see which changes directly impacted students. Later student evaluations noted the changes:

> "[The project] is a very good wrap-up of the whole year and should not be changed by much"

Rich course materials

For the university's *Computer Graphics I* course, the instructor only targeted HTML output and was able to embed interactive Javascript tools directly into the `CourseUp` lessons. An interactive RGB color selector code (Figure 10) was embedded in the lesson on color (Figure 11) and the tool was used in a quiz question on color values.

Sharing and reuse

The instructor for the university's *Computer Graphics II* course was able to reuse materials from the *Computer Graphics I* course, simply by copying the `CourseUp` files. A

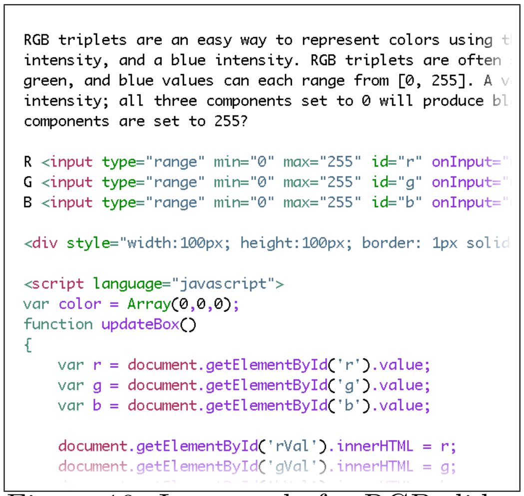

Figure 10: Input code for RGB sliders

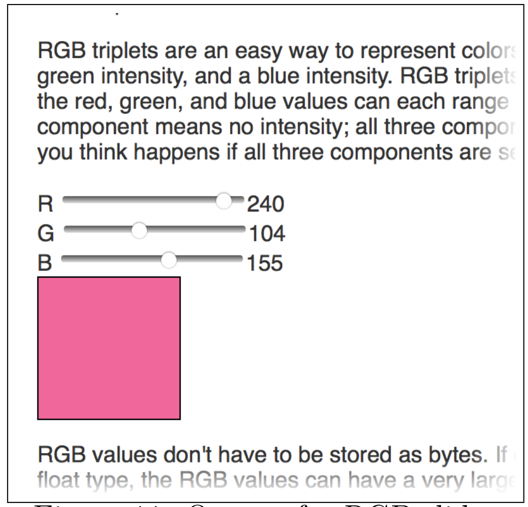

Figure 11: Output for RGB sliders

C++ tutorial was reused and updated for the more advanced course. Since all changes were tracked in a source repository, the *Computer Graphics I* instructor can easily decide if they wish to accept the modifications for use in their course.

Automation

`CourseUp` was used at the university for almost three years and nearly all of the course schedule management was automated. The courses were offered with a wide variety of term schedules. Each offering was adapted to the term by updating the handful of configuration lines defining the term schedule (see Figure 2). All assignment and exam due dates were derived relative to the generated course calendar and required no instructor management.

Performance

While `CourseUp` is a language definition, it is necessary to convert the `CourseUp` materials into a more presentable form for student use. The university has been using an HTML converter to output the materials. During a recent offering, the university's main web servers suffered an outage. Since `CourseUp` courses are relocatable, all `CourseUp` courses were migrated to a backup server with a simple file copy. We timed the performance of serving `CourseUp` documents as HTML on the primary server (an Intel Xeon E5) and the backup machine (an Intel i3 desktop PC). Performance was measured with HTTPS and HTTP keep-alive enabled.

	Concurrent clients				
	1	5	10	20	40
Server	15	17	27	47	130
Desktop	26	46	96	167	343

Table 1: Conversion and delivery in milliseconds

The results in Table 1 measure conversion and delivery, and do not account for the overhead of importing converted assets into an LMS. Nonetheless, we believe that this level of performance indicates that `CourseUp` materials could be hosted on commodity hardware, suitable for smaller institutes without access to dedicated high performance servers.

LIMITATIONS AND FUTURE WORK

There are several limitations when using `CourseUp`. The primary one is that there are no authoring tools for the language. This is acceptable for instructors with programming skills, but wider use would demand that better user facing tools be developed (similar to LaTeX and HTML WYSIWYG tools). In the future, we would like to adapt some existing user-friendly editors for use with `CourseUp`.

We are still developing the language definitions for online course development. While some useful components are defined (i.e. inputbox), there is much to be completed. We will continue addressing these needs by adding support for online concepts such as forums, surveys, and gradebooks.

Additionally, we have not developed output modules for popular LMSs. Many instructors wish to continue using their institute's LMS to present course materials. A converter would allow instructors to share, design, and track changes using `CourseUp`, but still present in an LMS. In order to accommodate this use case, we plan to create output modules that target Moodle and Blackboard.

CONCLUSION

We believe `CourseUp` is the first step in creating a robust and reusable course design language. By using best-in-class tools like Markdown, YAML, and LaTeX, we allow designers to create at their preferred level of detail, while retaining human readable qualities. We believe `CourseUp` accomplishes our goals of human readability, location flexibility, comparison support, and automation. The full `CourseUp` specification is available online at `http://courseup.org`. Example course definitions are also available.

REFERENCES

[1] Stina Bridgeman, Michael T. Goodrich, Stephen G. Kobourov, and Roberto Tamassia. SAIL: A system for generating, archiving, and retrieving specialized assignments using latex. *SIGCSE Bull.*, 32(1):300–304, March 2000.

[2] Tim Chen, Leonid I Ananiev, and Alexander V Tikhonov. Keeping kernel performance from regressions. In *Linux Symposium*, volume 1, pages 93–102, 2007.

[3] Hamish Coates, Richard James, and Gabrielle Baldwin. A critical examination of the effects of learning management systems on university teaching and learning. *Tertiary Education & Management*, 11(1):19–36, 2005.

[4] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[5] John Gruber. Markdown. `https://daringfireball.net/projects/markdown/`, 2004.

[6] ADL Initiative. Advanced distributed learning initiative. `http://adlnet.gov/projects`, 2016. Accessed: 2018-01-07.

[7] Christian Süß, Burkhard Freitag, and Peter Brössler. Meta-modeling for web-based teachware management. *Advances in Conceptual Modeling*, pages 360–373, 1999.

[8] Mulembwa Munaku Hashim Twakyondo and Mulembwa Munaku. Experience of course migration from blackboard to moodle LMS–a case study from UDSM. *International Journal of Computing and ICT Research*, 6(2):33–45, 2012.

[9] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[10] Kelly Wainwright, Mike Osterman, Christina Finnerman, and Bill Hill. Traversing the LMS terrain. In *Proceedings of the 35th Annual ACM SIGUCCS Fall Conference*, SIGUCCS '07, pages 355–359, New York, NY, USA, 2007. ACM.

[11] Richard E West, Greg Waddoups, and Charles R Graham. Understanding the experiences of instructors as they adopt a course management system. *Educational Technology Research and Development*, 55(1):1–26, 2007.