On Optimal Strategies for the Development and Operation of Moodle in Higher Education Institutions

M Omar Faruque Sarker, Jo Matthews, Jessica Gramp

Information Services Division, University College London, United Kingdom
f.sarker@ucl.ac.uk, jo.matthews@ucl.ac.uk, j.gramp@ucl.ac.uk

Abstract

Moodle has been chosen by thousands of higher education (HE) institutions to facilitate quality e-learning for millions of learners. This paper introduces the full end-to-end life-cycle of Moodle development and operation in a typical HE institution. This paper also discusses the strategies to overcome the key technical challenges related to the development and operation of Moodle in HE institutions. The important challenge addressed here is how to cope with the continuous upgrade process of Moodle to new major and minor releases. With the help of a few bespoke agile processes, this paper presents two case studies of upgrading Moodle to newer versions, from 2.2.7 to 2.2.9 (minor version upgrade) and 2.2.9 to 2.4.3 (major version upgrade combined with the annual Moodle archive site setup).

Keywords

Moodle, e-learning, learning management system, virtual learning environment, systems development and open source software maintenance.

Introduction

Higher education (HE) institutions require a cost effective and sustainable online learning management system (LMS) which can maximize the learning and teaching experiences of participants and, at the same time, minimizes the development and operation cost. Unlike many commercial proprietary learning management systems, Moodle excels as an open-source, cost-effective and community supported LMS solution. This paper discusses the strategies to overcome the development and operational challenges in maintaining Moodle in a HE institution. This paper is not about general Moodle development challenges. Rather it takes an institution's point of view into account to address the every-day challenges of managing the Moodle platform.

Given an initial setup of a Moodle production environment, there are several challenges to keep the Moodle platform up-to-date with the latest version of source code. Every new release of Moodle comes with many enhancements of existing features and, notable security and bug fixes. Institutions also add in-house and/or third-party developed plug-ins in order to extend the core functionalities of the Moodle platform. So, due to these obvious reasons, institutions can not overlook the necessity of continuously upgrading Moodle. However, like any other production software system, the Moodle platform cannot be upgraded without proper planning and preparation. This paper focuses on some key strategies which can be adopted by HE institutions for executing a smooth Moodle upgrade plan, e.g. when to decide to upgrade, how to plan and prepare for that and how to perform the upgrade systematically.

The rest of this paper is organized as follows. In the next section we outline the people, systems and tools involved in managing Moodle in a typical HE environment. Then we discuss the suitable development strategies following a systematic eight-week development and deployment model. In the next section, with the help of two case studies we describe our experiences of a Moodle upgrade following the previously discussed model. We also highlight a few operational strategies and a review of related works. Finally, we conclude this paper by discussing some key points and lessons learned from our case studies.

People, Systems and Tools around Moodle Application

The adoption of Moodle as an e-learning application in a HE institution creates an IT service ecosystem. There are certain roles involved in this ecosystem as outlined in Fig. 1.

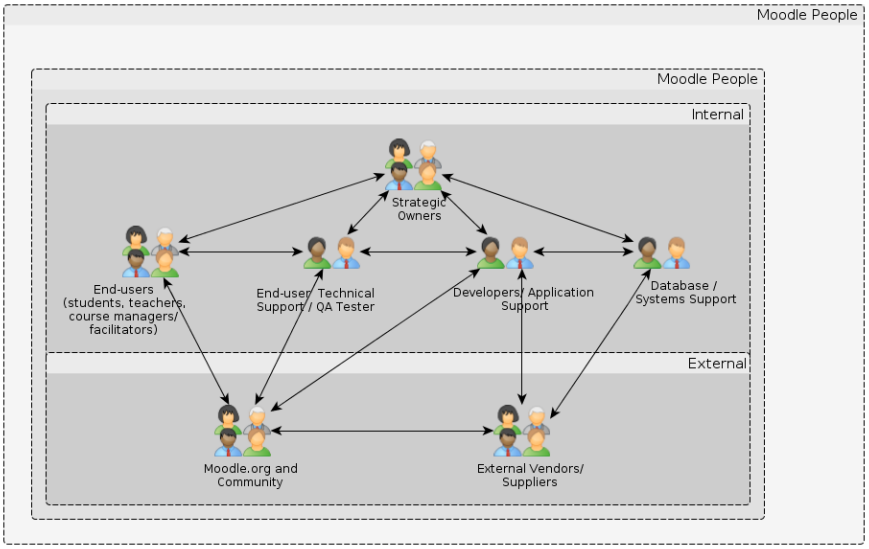

Figure 1: Roles involved in Moodle application service ecosystem

- *End-users*: Those who actually use Moodle, for example, students, teachers, course administrators, learning technology facilitators and so on.
- *End-user technical support*: Those who support the end-users with their day-to-day use and course level administration such as assignment submissions, grading, course backups etc.
- *Developers*: Those who develop Moodle plug-ins or customize themes and maintain the Moodle code-base. They take care of the institution's Moodle codebase, application upgrades, tweaking and maintenance. They also provide 2nd and 3rd line technical support to the end-users.
- *Database and systems support*: Those who are typically highly skilled technical IT staff and take care of the Moodle's server side administration, e.g. the operating system and database administration.
- *Strategic owners*: Those who sit above all the other roles and are responsible for resourcing e-learning in all aspects, for all strategic decisions on upgrades, and on stimulating effective use of the LMS.

The above roles regularly interact with two external groups of people. The first group, *Moodle.org and community*, has a large number of individuals who work on Moodle mostly voluntarily. The second group, *external suppliers and vendors*, include the official Moodle partners and other companies and individuals who provide various commercial services. Both of these groups are facilitated by the Moodle core team, also known as the Moodle headquarters (HQ) team.

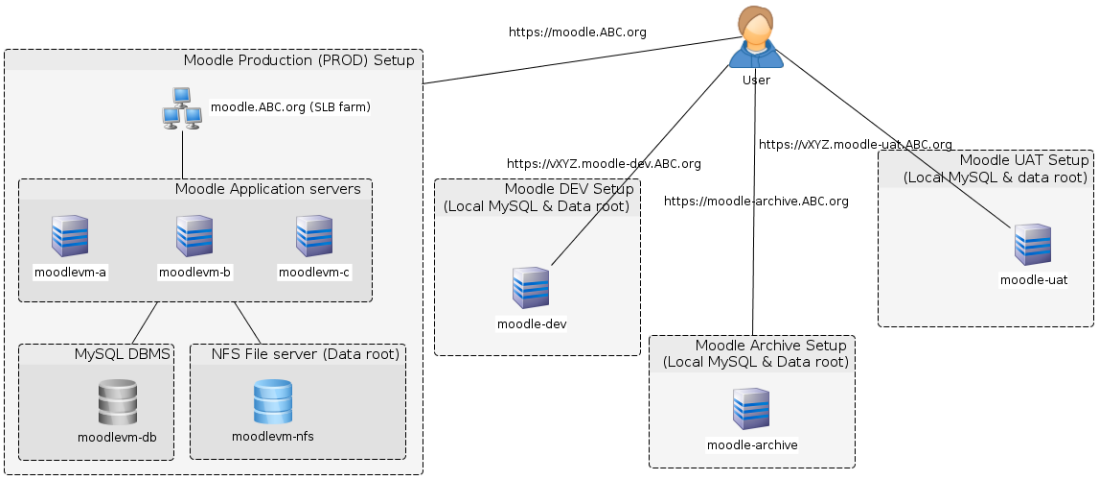

Figure 2: Moodle application environments in a typical HE institution

The sustainable and reliable deployment of Moodle relies on the systematic approach of managing this application. This includes setting-up different application environments for development and operation, as shown in Fig. 2. We categorize the many types of Moodle systems aka *environments* into five major groups.

1. *Moodle Production (PROD) environment*: This environment contains the live Moodle *instance* serving the end-users. The major systems around this instance include: a load balancer, few application servers (three are shown here: *moodlevm-a/b/c*), one dedicated MySQL database server and one NFS file server that hosts Moodle application data.
2. *Moodle User-Acceptance Test (UAT) environment*: This environment contains at least two Moodle instances: an exact clone of the current production instance and a previous version of the production instance. There are certain reasons to justify this redundant setup. For example, if any bug is reported, developers can try to reproduce that bug in the UAT environment's current production clone and the previous production clone. Similarly, any fix of any bug or new feature must be tested in the UAT instance before rolling out to the production environment.
3. *Moodle development (DEV) environment*: In this environment, several Moodle instances can be hosted, each based on a specific Moodle version and a set of new features or bug fixes.
4. *Moodle archive (ARCH) environment*: This is another live environment which contains read-only instances of previous academic years' clones of production instances. For example, if the current academic year is 2012-13 the ARCH environment should host the read-only production clones of year 2011-12, 2010-11 etc. These instances are kept for future reference and for complying with data protection legislations.
5. *Moodle local (LOCAL) development environment:* This environment sits on each developer's working machine. It contains an exact or a modified clone of the production instance.

Several tools or platforms are necessary to facilitate the interaction among the Moodle people and systems. Some of them are listed in the following table.

Table 1: Tools that facilitate Moodle application service

Issue Tracker	End-users, or end-user technical support staff, open new issues as tickets in the issue tracking system. Developers post updates on these tickets.
Version control System	Developers use a Version Control System (VCS) to keep track of source code changes during the life-cycle of the Moodle application. Each code release on each different Moodle environments e.g. UAT/PROD is tagged for future reference and comparison.
Service Monitor	A service monitoring tool is essential to alert developers and system support staff of any service outage or server faults. The systems support team manages this tool and configures it to send alerts.
Wiki	A wiki is also a recommended tool in order to store documents and to share the common discussions and knowledge amongst the various groups of people (see Fig. 1).
Change Logging System	A change logging system is very helpful to communicate production changes with higher level technical and management people. For example, each production code change can be documented to make it clear why the change occurred and how to roll it back should any problems arise in future.

Development Strategies

The core Moodle software is developed and maintained by the Moodle HQ core development team. As Moodle has a very flexible plug-in based architecture, third-party plug-ins written in a specified way can be incorporated into Moodle core code. This will extend and enhance the core functionalities of Moodle. In this section we describe a development model with some strategies to manage a HE institution's Moodle codebase.

Moodle Codebase

As seen in Fig. 3, Moodle's code-base can originate from three sources: the *core code* from Moodle HQ team, third-party plug-ins from the community and in-house code developed at that institution or by external developers. The latter two types can be categorized as *non-core code*. There are many types of non-core code. For example, *themes* customize the look and feel of a Moodle site and *blocks* are user interface components that can be added to any page of a Moodle site.

The core Moodle code is released in major and minor versions. For details about the version numbering, release labelling and, code branches the reader should consult the Moodle community website (Moodle Version, 2013). Any institution installing Moodle starts with a major version e.g. *X.Y.Z* and then this code-base can be upgraded to the next minor version *X.Y.(Z+1)* every two months. On the other hand, in order to get all the latest stable features institutions can upgrade their Moodle to the next major version *X.(Y+1).Z* every six months.

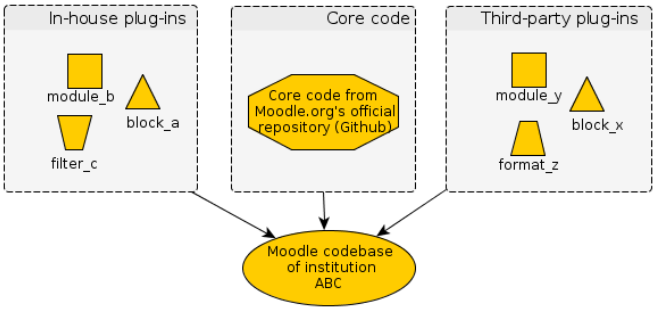

Figure 3: Composition of Moodle codebase at a HE institution

Development cycles

The Moodle development context can be visualized by comparing its usage at different times of an academic year. In a typical HE institution, if the academic year starts in September, we can set a typical pattern of Moodle development requirements throughout the year.

- A few weeks/months before the academic year begins, the necessity of a Moodle major version upgrade can be reviewed. The decision to upgrade to a major version can be influenced by the amount of security fixes present in the target release, end-user feature requests and bug-fixes present in the target release. The optimum time to do this may be just after archiving occurs at the end of an academic year. If the major upgrade is unsuccessful, the rollback can be done from the recent archive copy.
- After the major version upgrade, at the start of the academic year, developers can review the outstanding technical issues and development/feature requests from the previous year.
- During an academic year, there will be several Moodle minor versions released for a particular major version. New in-house and third-party plug-ins can also be added at this time. So a series of minor version upgrades can happen throughout the year.

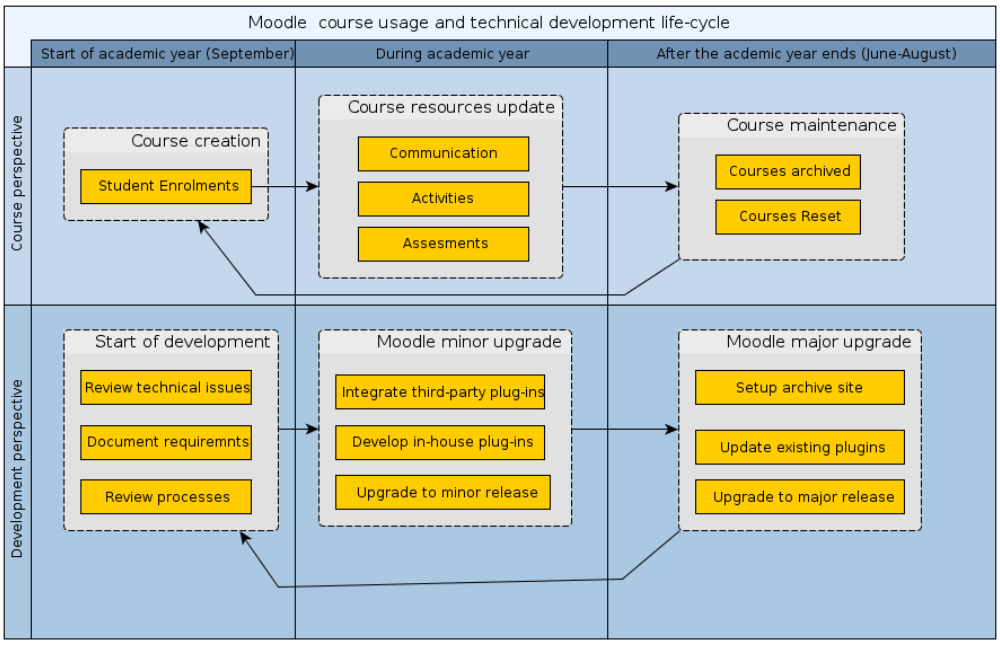

Figure 4: Moodle development contexts within the usage life-cycle

Minor upgrade cycle

The development milestones of a typical minor upgrade cycle is illustrated in Fig. 5. In the first stage of the development process, developers start fixing issues and developing new features in their LOCAL and DEV instances. These can briefly be checked by end-user technical support staff. At the end of this stage a decision is made regarding what tickets will be integrated with the next release and when to push them to UAT.

In the second stage, a UAT instance is prepared for a thorough integration testing. Firstly, a recent copy of the Moodle database is fetched from the production MySQL server and is loaded onto a test database on UAT. Then the release candidate code is placed onto a new webserver path and a Moodle upgrade script is run against the loaded database. After the upgrade is successful, QA testers and end-user technical support staffs start integration tests on this new instance. Sometimes some new features may not be properly integrated with other features. When these are spotted at this stage, QA testers can log them on the issue tracking system. Developers can concurrently work towards fixing these integration bugs. After some time, UAT code can be updated with new fixes and then final UAT testing can be completed. At the end of this testing, UAT code should be put into frozen mode where no new feature or fix is allowed, except for emergency security fixes.

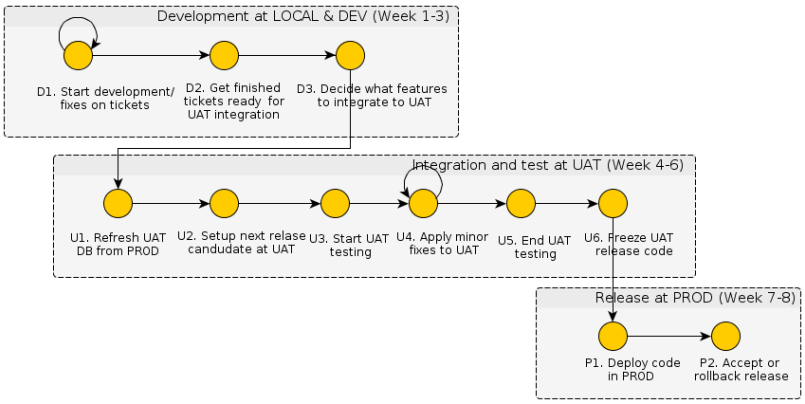

Figure 5: A typical eight weeks minor upgrade/development cycle

In the final stage, on a specific, pre-advertised date, the Moodle production instance is put into maintenance mode and the upgrade is performed. An additional week is reserved for settling the production environment. During this time, in case of any unexpected or regression failure, production code can be rolled back to the previous version.

Archive Creation

At the end of an academic year, usually HE institutions will be required to take a snapshot of the production Moodle site. Some useful archiving strategies are discussed here.

Setup a Moodle archive site: As outlined in Fig. 2, a Moodle archive environment contains multiple instances of previous years' Moodle sites. This can be achieved by setting up a single webpage that lists the links to all academic years and creating sub-directories for each year's Moodle codebase. In this way users can come to a single archive homepage and choose which year's archive they wish to explore. However, in order to prevent web browser session clashes due to multiple instances of the Moodle archive sites running on the same server, site cookie prefixes should be changed in the Moodle admin settings (site administration > server > session handling). Other changes to this site configuration should include removing guest login, disabling e-mail sending etc.

Use a different Moodle theme: End-users are likely to be confused with many instances of Moodle archives and with the production Moodle instance. In order to prevent this, it is necessary to use some visual indications that will tell end-users that they are viewing a particular Moodle archive site, not the production site. Using a different Moodle theme with a distinct banner or background image can help. Some other important

modifications include modifying the Moodle site name and changing the original site links within Moodle pages and activities to point to the archive site and not the production Moodle.

Modify the Moodle user roles: In an archive site, all students should be assigned to a read-only student role and all tutors should be assigned to a read-only tutor role. The capabilities of these read-only roles may vary depending on the Moodle version and the features used in a particular site. For the student read-only role it is also necessary to ensure that the graded role is only assigned to the read-only student role, otherwise students grading information may not be available. All existing students and tutors should be mapped to the newly created read-only roles by running suitable SQL queries.

Prevent enrolment in courses: The archive site must not allow any new students to enrol in any Moodle course. In order to achieve this, all enrolment plug-ins can be hidden in the Moodle site administration settings.

Backup the production database and application data: The Moodle production database should be backed-up prior to setting up the archive site. Database backup time can be minimized by setting up database replication between these servers. It is a good practice to copy the Moodle application data to a different physical media such as tape for long-term storage. In order to minimize the time to back up, an earlier full copy of application data can be created. So on the archive site setup day, an incremental backup can be performed in a short time.

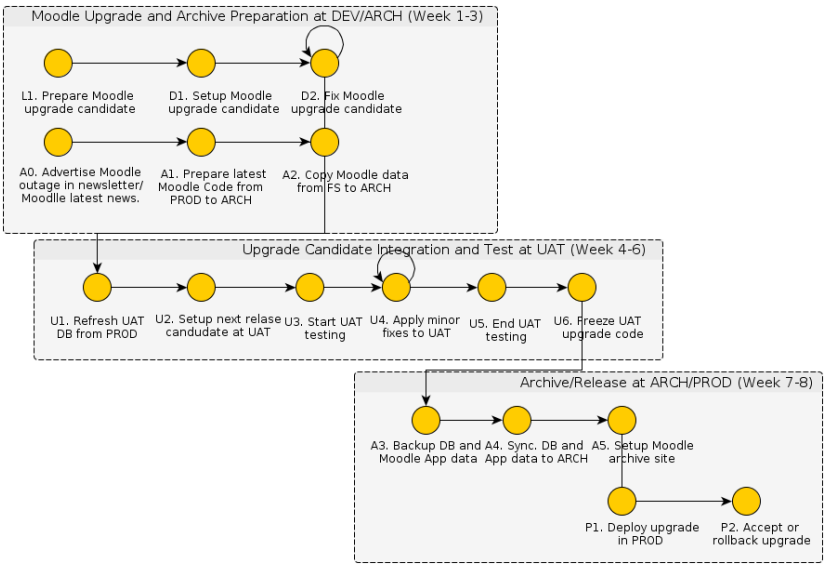

Figure 6: Moodle major version upgrade and end-of-year archive creation

Major version upgrade combined with archive creation

In a typical HE institution, we can anticipate that Moodle major version upgrades will be considered prior to the beginning of an academic year. This upgrade process can be combined with the end-of-the-academic-year archiving process.

As outlined in Fig. 6, a combined archive creation and major version upgrade process is shown here. This major upgrade process is similar to the previously described minor upgrade process apart from a few things. Firstly, it runs in parallel with the archive creation process and secondly, instead of developing new features, the development process is more concerned about testing compatibilities of the existing Moodle modules with new ones and updating the plug-ins following the upgrade guidelines from the Moodle HQ core team.

In the first stage of this process (steps L1, D1-D2, A0-A3 in Fig. 6), end-users are notified about the Moodle service outage during the upgrade and archive creation process. The exact duration of this outage can be found by simulating the entire process in a UAT instance. For the major upgrade, developers can set up a candidate upgrade release of the Moodle codebase. This can contain the Moodle core code, updated versions of all existing

plugins and any local changes to core modules ported from the current production code. A release-note, written as a plain-text file, can be very helpful to document the contents of this codebase. This candidate release is then setup in a DEV environment and tested by QA testers and tweaked by the developers. A new issue tracking system can be setup to log development issues. At the same time, the current Moodle production code and production database can be copied to an ARCH environment and tweaked for the archiving process as discussed earlier.

The second stage of the development process is similar to that of the minor upgrade cycle. But in the last stage of this process, it is advisable to ensure that an archive site is setup before deploying the upgraded release to the production environment. This helps the end-user by allowing them to continue using the archive site (the read-only copy of Moodle) while the upgrade is being performed. During this upgrade period users attempting to access the production Moodle site are redirected to the archive site.

Operation and Support Strategies

High availability and reliable performance are two of the key requirements of running a Moodle platform at a HE institution. Several strategies can be adopted to ensure development is possible, while maximizing the service uptime with desired performance. A 24/7 server monitoring and event alerting system should be setup with the help of a service monitoring tool. This can alert developers and systems support teams of critical server issues e.g. low disk-space and low memory, slow SQL queries or any component failure.

In a particular day of every week, a two-hour scheduled Moodle maintenance outage can be advertised to the end-users. During this maintenance period, Moodle upgrades or occasional security patches can be applied to the production server. However for an outage occurring during standard work hours, or one that is longer than 2 hours, end-users must be contacted to see if there is any potential impact e.g. scheduled online examinations or online submission deadlines. This strategy will reduce the communication overhead and at the same time will keep the end-users informed of any critical service upgrades. Any issue reported by the end-users, or end-user technical support team, should be logged in the issue tracking system for resolution and future reference. Communicating purely through e-mails or face-to-face conversations are not viable for tracking issues in the longer term, as these are not as easy to refer back to.

For the optimum security of Moodle instances, a set of security rules should be developed. Access to production servers must be kept limited to specific users. Separate user logins should be used for accessing all Moodle environments described in Fig. 2. This will minimize the risk of compromising the entire production environment by a leakage of a single user's login credentials. The systems support team are also responsible for keeping the operating system and application software stack up-to-date by applying the latest security patches.

Moodle Upgrade Case Studies

At our institution, the Moodle production instance hosts about 6,800 courses, used by over 72,000 users. The size of our end-user technical support and development team is 5 and 2 respectively. We have some systems and database support staff available on-demand. The structure of our Moodle environment is the same as shown in Fig. 2. Our Moodle instances are based on standard Linux/Apache/MySQL/PHP (LAMP) stack. We use the following tools for facilitating team interactions.

- Trac issue tracker: Trac provides an intuitive interface to add/edit tickets. For each major Moodle version upgrade, we create a new Trac project and store all tickets on that project.
- Git Source-code Version Controlling System: We use Git for Moodle 2.4.x based releases.
- We use a combination of Puppet, Nagios and Opsview system monitoring tools for 24/7 service monitoring.
- Confluence wiki: This is used for documentation and team collaboration.
- Other tools: For password management we use KeePass tool for storing all our login passwords.

Case Study 1: Moodle minor upgrade

The first stage of our upgrade process was started by creating an entry on our change logging system. After the change proposal was reviewed and approved through our change advisory board meeting, a release candidate tag was created in our Git repository. In this release tag, we updated the release notes reflecting what new features were added and what Trac tickets were included in this release. This selection was finalized by consulting with the end-user technical support team. In our case study, this first stage was launched on the beginning of March 2013 and ended after three weeks. A total of 8 Trac tickets were included in this upgrade.

In the second stage of our upgrade process, a UAT instance was created in the fourth week. QA testers started testing and test packages were signed-off at the end of the first week of April 2013. UAT code was frozen in the subsequent week and then the production environment was prepared.

At the last stage of the upgrade, the entire upgrade process was reviewed before carrying out the final production roll-out. This was done by consulting the Moodle official release notes, upgrade instructions and experiences gained earlier through setting up the DEV and UAT instances. On the day before upgrade, we put the release candidate code on the application servers. The following list highlights the major steps involved in deploying a new upgrade in the production environment.

- Disable end-user access: We put Moodle into maintenance mode and disabled access to production instances from the outside world except for a few local test machines through Apache configuration.
- Backup the MySQL database: After we stopped the CRON job on the main application server the Moodle database was backed up.
- Run upgrade: After enabling the next release candidate code on the production instance, we invoked the Moodle upgrade script from the command line. Then QA testers were invited to test the new instance briefly.
- Resume service: After the test was confirmed successful, we started the CRON job and restored access to all webservers from the outside world. We disabled Moodle maintenance mode and emailed all teams about the completion of the upgrade process.

Case Study 2: Moodle major upgrade and archive creation

After the minor upgrade from version 2.2.7 to 2.2.9, a decision was made to upgrade to Moodle 2.4.3 within eight weeks. This upgrade was also done in conjunction with the archive creation of the 2012-13 academic year. Two developers started these two tasks in parallel. A new Trac project and a Git repository were setup. Two institution specific branches were setup e.g., *uclmaster* and *ucldevel*. The uclmaster branch was initialized after cloning code from the Moodle core repository hosted at Github. All experimental code developments were done on our ucldevel branch. When a release candidate was ready in the ucldevel branch it was merged with the uclmaster branch and tagged for releasing to the DEV and UAT environments. An update from Moodle core code was also picked up by the institution-master branch for fixing some relevant issues.

In our new issue tracking system, about 30 tickets relating to the upgrade were logged by the testers. Most of these issues were the result of an incompatible theme and local changes made to the core code in the earlier minor upgrades. The lesson learnt from this case was to minimize the local changes to core code as much as possible. In this upgrade release, all valid Moodle 2.2 assignments were upgraded into the Moodle 2.4 compatible version and some database schemas were also updated. After finishing the first cycle of testing on the UAT instance, testers reported about 5 integration issues and they were fixed in the second iteration. At the end of this, the UAT code was kept frozen for a week and the final upgrade code was deployed on the production server.

The 2012-13 academic year's archive site was setup three weeks prior to the upgrade day. All application data was copied to the ARCH environment one week prior to the upgrade. On the actual upgrade day the production Moodle site was put into maintenance mode and the production database was backed up. The redundant application servers were shutdown. Another incremental backup of application data was taken by *rsync*. After the archive site was setup, the main upgrade process was started and lasted for a couple of hours. When the testers were happy with the upgraded site, all Moodle application servers were brought back to normal service

and the Moodle site was taken out of maintenance mode. This upgrade exercise was started on Friday afternoon at 5pm and took two and a half days to finish.

Related works

A lot of research, related to the selection and usage of Moodle as an e-learning platform, has been reported in the literature. For example, Beatty and Ulasewicz (2006) reported their experiences of migrating from Blackboard to Moodle from a faculty's perspective. A lot of books, user manuals and multimedia tutorials have been published on Moodle from an end-user's perspective (e.g.as a student or a teacher). A list of books can be found on the Moodle community website (Moodle Books 2013). Some of these books are written to help Moodle administrators and developers to do their jobs better e.g. for theme design (Gadsdon 2010) and for general Moodle configuration and administration (Buchner 2011). The Moodle HQ team maintains the development-process documents on the Moodle community website. However, these documents have nothing to do with optimising the development and operation of Moodle in HE institutions. This paper tries to fill in that gap to some extent.

Conclusion

In this paper, we have presented a set of strategies related to the development and operation of the Moodle platform from a HE institution's points of view. These strategies aim at ensuring the minimum disruption of the Moodle service while enhancing and maintaining the core platform. We have also briefly presented our experiences of upgrading our Moodle production instance and creating an end-of-the-academic year Moodle archive site. A few lessons learned from these case studies include: advertise Moodle service outages as early as possible, document the core code changes very well and remember to disable the service monitoring and alerting during the upgrade process.

References

Beatty, B., & Ulasewicz, C. (2006). Faculty perspectives on moving from Blackboard to the Moodle learning management system. *TechTrends*, *50*(4), 36-45.

Buchner, A. (2011). *Moodle Administration: An administrator's guide to configuring, securing, customizing, and extending Moodle*. Packt Publishing.

Gadsdon, P. J. (2010). *Moodle 1.9 Theme Design: Beginner's Guide*. Packt Publishing Ltd.

Moodle Books. (2013). Retrieved June 3, 2013, from https://moodle.org/mod/data/view.php?id=7246

Moodle Dev Process. (2013). Retrieved June 3, 2013, from http://docs.moodle.org/dev/Process

Moodle Versions. (2013). Retrieved June 3, 2013, from http://docs.moodle.org/dev/Moodle_versions