REGIS: A Tool for Building and Distributing Personalized Practice Problems

Albert Segars
Dan Garcia, Ed.
Dawn Song, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-101
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-101.html

May 11, 2012

REGIS: A Tool for Building and Distributing Personalized Practice Problems

by Albert Lucas Segars, III

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Lecturer SOE Daniel Garcia
Research Advisor

(Date)

* * * * * * *

Professor Dawn Song
Second Reader

(Date)

ACKNOWLEDGEMENTS

First and foremost I am thankful to Chris Cartland and Katrina Chang for their work and vivid insights as co-developers of this project. The time they've volunteered has significantly improved the direction of the project and made it into far more than it would have been otherwise. Both Chris and Katrina put in a tremendous number of hours brainstorming, mocking up, developing, and debugging and I cannot thank them enough for their time and effort.

I'm also extremely grateful for the mentorship and guidance from my advisor Dr. Dan Garcia, who has been a tireless supporter, motivator and friend throughout my entire graduate experience. He helped me understand how much of a difference high-quality educators could make on student lives and has been one of the key inspirations for my interest in improving the state-of-the-art in education. He's also helped me set high expectations for myself with my teaching, which quickly became (and, until the end, remained) my favorite part of my experience at Berkeley.

All of my coworkers and friends on the CS10 team deserve a huge thank you as well. I have honestly never been on a team with so much passion, persistence and enthusiasm for what they do and it's been one of the greatest honors of my life to work with them. Thanks for your feedback on this project as well as the many lessons you've taught me along the way.

ABSTRACT

A wave of web technologies are emerging that allow students to receive immediate feedback, discuss topics with instructors, and learn from their peers. Many of these tools offer improvements over the traditional classroom experience, but very few provide effective ways for students and instructors to connect with each other over domain-specific knowledge as a means of engaging with the material.

REGIS is a tool for developing and distributing customized problems to groups. The system has been designed primarily for college-level students, but is intended to be general enough to apply in a variety of other settings as well, e.g. K-12, informal study groups, and online courses. Instructors and students are able to create questions, submit answers, instantly provide feedback to students, and receive measurable results about student learning and comprehension. The system uses an interface based on a flash card analogy and is available online under an open source license.

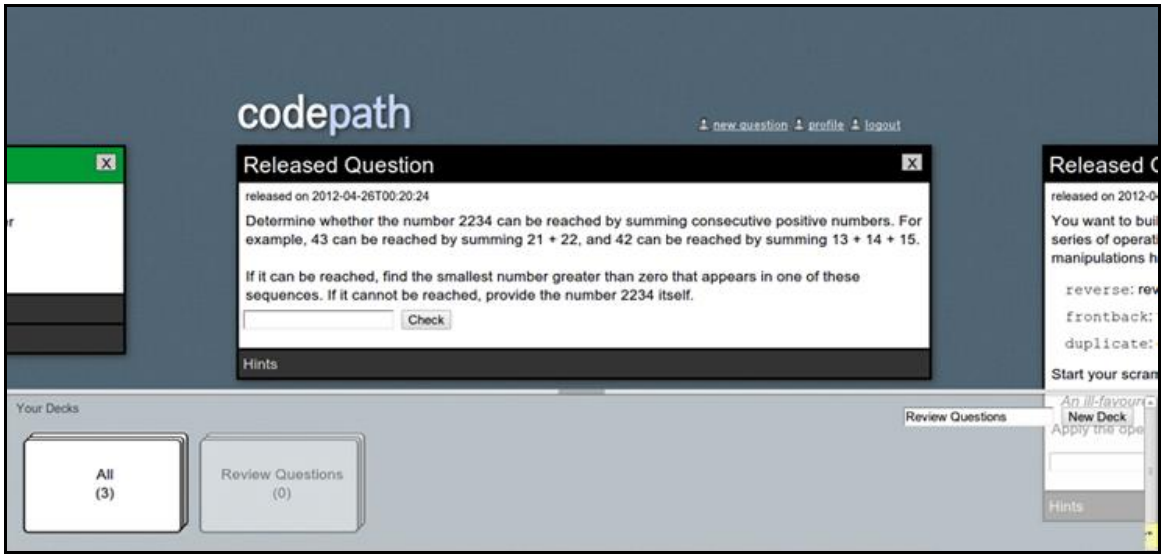
We have developed REGIS for use in our non-majors computing course at UC Berkeley and launched the software to our students during the spring of 2012, monitoring usage statistics and gathering student feedback at the end of the semester. Initial feedback suggests that students are interested in using REGIS to help them practice, but may need a more structured means of interacting with the system than we provided in our trial.

TABLE OF CONTENTS

INTRODUCTION

Opportunities to apply knowledge can be critically important to developing confidence, interest, and a clear mental model for a topic [1]. Despite its importance, personalizing opportunities that develop an in-depth understanding and appreciation of material is challenging for instructors in many-to-one teaching scenarios to provide due to the time requirements of generation and implementation. This bottleneck limits the effectiveness of the traditional educational process as well as the learner's ability to absorb the information being presented to them. The recent rise of online learning threatens to exacerbate this bottleneck due to drastically expanding student-to-faculty ratios. The hope is that innovative new technologies can mitigate the problem.

REGIS (Random Exercise Generation and Inference System) is a web-based tool for authoring and distributing customized problems or activities to groups of individuals. We have developed a simple language that can be used to author questions and variations on each question can be easily generated for arbitrarily large pools of users (students, workshop attendees, study groups, etc). The system presents these questions as a series of flash cards and can either evaluate the correctness of responses automatically or through a manual instructor- or peer-review interface. Figure 1 shows a screenshot from a live deployment of the system.



released on 2012-04-26T00:20:24

Determine whether the number 2234 can be reached by summing consecutive positive numbers. For example, 43 can be reached by summing 21 + 22, and 42 can be reached by summing 13 + 14 + 15.

If it can be reached, find the smallest number greater than zero that appears in one of these sequences. If it cannot be reached, provide the number 2234 itself.

Figure 1: This screenshot shows REGIS's primary user interface. The questions are presented as flash card-like elements, and the users question collections, or *decks*, can be easily accessed through the dock at the bottom of the screen.

We have developed the tool primarily for an introductory university-level computing course at UC Berkeley [2], but the design is broad enough that it should be generalizable to a variety of disciplines and contexts. After receiving feedback suggesting that such a tool would be useful, we launched a deployment of REGIS in our course and summarize the participation characteristics that we observed and design choices that we made. The goals of the project are in line with the goals for our course: to provide both a comprehensive understanding of the power and complexities of programming while also developing an intrinsic interest and appreciation for the applicability of computing to solve a wide range of different problems.

In addition, we want REGIS to minimize the administrative time and effort spent generating practice programming problems, distributing them to students, keeping such lists up-to-date, and confirming the correctness of solutions. Our system makes it possible to automate and simplify these tasks and simultaneously increase the availability of practice resources for students. In addition, comprehensive analytics can be gathered and presented to administrators to gauge student understanding and interest for particular topics.

Target Audience and Design Requirements

This project was designed for a computing course for non-majors at UC Berkeley. The course, titled CS10: The Beauty and Joy of Computing, involves a significant amount of programming and is targeted at students with no previous experience in the field. These are either non-majors or computing majors who need to fulfill the programming prerequisite for our introductory computing course. The course also covers an array of social issues related to technology in addition to programming concepts.

In the fall of 2011 and spring of 2012 we surveyed our students in CS10 (n = 173, 168) during lecture in an attempt to determine what additional tools or support could be added to the course to ease the learning curve and help further engage students in the material. In addition, we asked about tools that could be useful for an online version of the course, which we are currently in the process of developing. The anonymous surveys contained five different choices for the students, which are listed in full in Appendix A and summarized in the chart below.

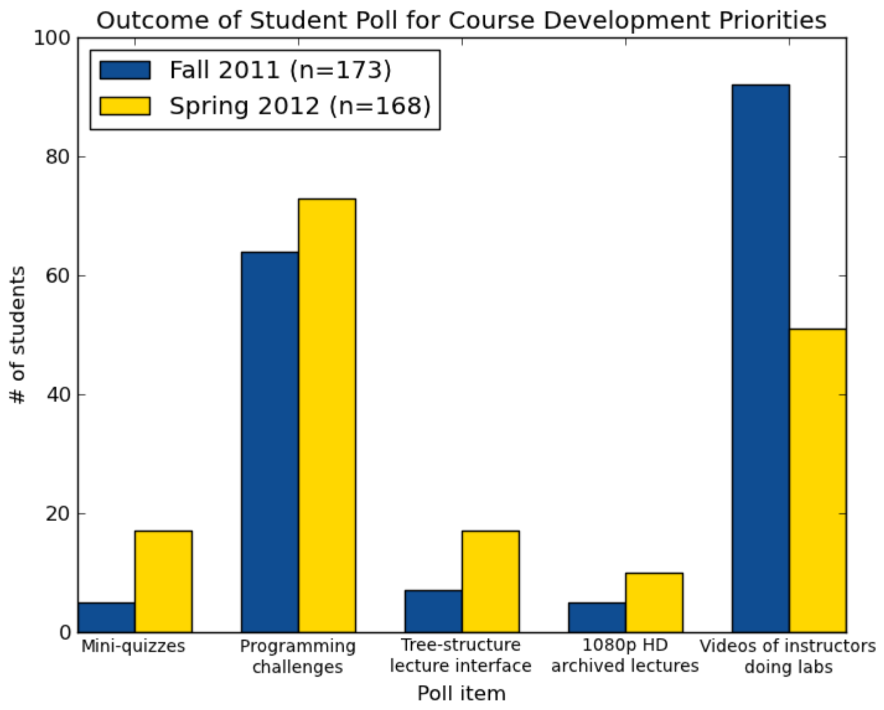


Figure 2: The results from our polls of CS10 students during fall 2011 and spring 2012. They had a clear preference for programming challenges and videos of instructors doing the labs, homework assignments, and exams. REGIS was designed to provide the programming challenges. Full tables are available in the appendix.

Both surveys produced very similar results; the top requests among students for both semesters were for mini-programming exercises (which REGIS aims to address) and instructional videos. In both cases, these two options received significantly more votes than the other available options. We announced REGIS several weeks into the spring 2012 semester of the same programming course. In the *Preliminary Results* section we will discuss adoption and usage patterns for REGIS in our course. We used REGIS as a supplemental tool that had no impact on student grades because we were interested in seeing what usage patterns would emerge when students were given the option of participating voluntarily. This also gave us the opportunity to see if there was any correlation in test scores among those who chose to use the tool and those who did not.

Our course has been designed around a lab-centric curriculum [3] and contains a significant number of structured exercises that are designed to teach students about fundamental programming concepts. We designed REGIS to distribute challenging and unstructured exercises that provided opportunities for students to apply their skills by exploring interesting problems.

A number of design decisions were influenced by our target audience, although we aimed to make the tool general enough to apply to a range of circumstances including K-12 classrooms, academic competitions, and exam review sessions, in almost any topic of study. Some of the major design decisions are summarized here, and more information is available in the *Mechanics and Major Design Decisions* section below.

First and foremost, we wanted the tool to be straightforward and supportive whenever possible; the question sequencing engine, responsible for determining which students see which questions, will quickly drop a user's estimated skill level if they begin missing questions in order to provide them with an opportunity to boost their confidence. It is also possible to provide customized feedback to common mistakes that are made on particular problems. Students can contribute their own problems to the corpus of available questions and share them with other students. We have redesigned the interface several times in order to remove distractions and focus on the user's intended goal of creating and answering questions.

Second, our course uses a graphical programming environment that has far slower runtime performance than most modern languages. Questions were selected that would be computable in reasonable amounts of time ($\leq$ 30 minutes, often far less) in our programming environment. Many common programming tools (iteration, recursion, first-class functions, etc.) are available in the language and questions were added that covered a broad range of techniques when possible. Often students outgrow the graphical language by the end of the semester and want to experiment with other languages, often including Python, Java, and PHP. We chose to make REGIS language-agnostic in order to make the tool as broadly useful as possible, both for our class and for other external use cases.

Finally, we wanted the tool to be flexible enough to be used for anything from a single exam to a multi-year learning experience. The tool should work in both formal (classroom) and informal learning environments. This means that we avoided using structures and diction that might imply that the learning is taking place in a classroom, and instead built on top of a flashcard analogy that allows each question, or "card," to be grouped in arbitrary ways with very little administrative overhead. Groups of individuals, organized in REGIS as "leagues," can be constructed and destructed very quickly.

RELATED & PRIOR WORK

REGIS shares many ideas with several fields of preexisting work from educational psychology, HCI and machine learning. We are unaware of other projects that use the flash card metaphor to target higher level learning beyond memorization, and that allow teachers and students to work together to build answerable sets of questions for reviewing and learning new ideas. Several of these ideas have been addressed individually and are outlined below.

Pedagogical Motivations

It is well-recognized that practicing a skill leads to an improvement in performance. In ideal circumstances, individuals (teachers or other students) would provide effective practice problems and students could practice with them at a convenient time. Both the generation and solving of problems tend to be relatively time-consuming activities, and REGIS aims to lessen the burden for both activities and allow users to focus on the learning experience instead of the configuration and administration of a complex system.

In addition to our goal of improving student comprehension and comfort with programming and problem solving, we also wanted to help develop an intrinsic interest in the field among users who may not be motivated or informed otherwise. Developing interest is both a system-level user experience challenge and a question selection challenge.

Development of Interest and Intuition

It is not surprising that having a high degree of interest in a topic often makes learning about that topic more enjoyable and, in most cases, more effective. Mechanisms that inspire both short- and long-term interest can fortify an individual's confidence and self-efficacy [4]. These mechanisms can also influence an individual's selective persistence, or willingness to focus on one option above others due to a high interest level. Both the motivation of interest and selective persistence will be discussed further in this section.

Deci and Ryan's popular self-determination theory [5], [6] describes both intrinsic and extrinsic motivational forces that impact human behavior. Extrinsic motivators, which utilize evaluations, tests, extra credit, career talks, and other forms of forces that involve "separable reinforcement," are commonly used in academic settings. Intrinsic motivators, which are those that occur in the absence of external rewards, are harder to foster and implement but can lead to higher levels of enjoyment and creative expression in the activity [5]. Interestingly, researchers have identified perceived control [7], competence [8], and autonomy [9] as the three inherent psychological needs that are most closely related to one's level of intrinsic motivation. In an educational setting, both are particularly important when learners are trying something that they consider to be new, and it is important that one's feelings of competence and autonomy not be overrun in order to preserve the desire to learn. Providing rigid assignments that are well beyond a student's skill level is one common way that a student's motivation is often diminished.

Developing an interest in a topic can therefore motivate individuals to focus on learning more about it. Selective persistence is a term that describes the correlation between the intensity of one's interest with both the persistence of that interest over time and the likelihood that one will engage in the activity without external contingencies (extrinsic motivational forces) [10]. As the intensity of an individual's interest in a particular object or idea increases, the likelihood of developing a long-lasting intrinsic interest increases. The development of interest is incredibly important in the realm of education; motivating students' intrinsic interests makes the education process far more rewarding and productive.

Content personalization offers a means of customizing lessons to minimize the loss of motivation due to feeling lost, confused or incapable of completing a task.

Interactivity & Autonomy

Traditional classrooms can easily become one-way pipelines where instructors provide information to students and students hope they receive it in an understandable format. Opportunities for practice and evaluation are notably guilty of this; the majority of practice opportunities come from instructor-provided homeworks, labs, tests, in-class activities, and other forms of assignments. One of our goals with REGIS is to make it easy for users, in this case formal students, to provide questions and to see each other's responses as well.

Our course attracts students from a wide variety of backgrounds with a range of interests. Providing exercises that are interesting to all students is challenging from an instructor's perspective due to the wide range of domain knowledge that is required in order to capture the curiosity of many students. Students or other authorized contributors with knowledge in other domains, however, could likely do this with very little effort. User engagement metrics can then be used to determine which of the contributed questions are most interesting to users in order to reduce noise and provide recommendations. All users therefore have the ability to become both content producers and consumers, either of which improves the experience for others.

Another notable advantage to hosting a diverse range of questions is that it provides users with an opportunity to discover and share problems that they find to be interesting or challenging. Students can then discover applications or problems that they were previously unaware of and pursue them if they seem interesting. Cross-disciplinary applications of knowledge, for example, could be shared by those who pick up on them.

Practice and Flash Cards

Physical flash cards are often used for memorization tasks, but this is a restriction of the format (the physical card), not the interaction. The flash card metaphor, used occasionally in other digital systems like Apple's once-popular Hypercard product, provides a series of pedagogical benefits to the user when used in an academic setting:

- The prompt and the answer itself are available in a very tight feedback loop: the front and back of a single card.
- Flash cards can easily be arranged, grouped, and categorized by moving and stacking them together. Cards can also be added or set aside as their relevance varies.
- Iterating over problems presented in flash cards is trivial.
- Flash cards can be used effectively both as an individual and with a group.

As mentioned, the historic use case for flash cards has been primarily for memorization tasks and has shown a tendency to improve short-term retention significantly when used in memorization-based exercises [11]. REGIS is able to support memorization-based tasks, but also aims to support higher-level learning tasks as well; the methods for doing so will be the primary subject of the remainder of this paper.

Digital flash cards

As mentioned previously, work at a variety of academic levels [11], [12] suggests that flash cards can be an effective method for learning due to the tight feedback loop between providing an answer and receiving an assessment, particularly with memorization-based tasks. Schmidmaier et al designed a set of digital flash cards for medical students and found that short-term recall improved among students who used flash cards for memorization tasks but long-term (>6 months) recall was not impacted.

Several studies, including those from the work of Tan & Nicholson [12] and Stutz [13] suggest that younger students report that flash cards can be an effective and fun way to learn new information, in this case vocabulary words in a foreign language. Stutz actually suggests having students create their own flash cards to share with their peers in order to reduce the workload on the teaching staff and provide the students with an opportunity to express their creativity.

It is important to note that these projects used flash cards in the context of fact memorization as opposed to higher-level learning objects like application or synthesis; REGIS aims to make it possible to do the latter as well.

Evolution of "school"

Individual access to information has increased significantly with the popularization of the Internet and related technologies; many people are beginning to rethink the idea of the traditional classroom and how it can best be adapted to an age when almost all information is already available to the students through other means. Many educators and companies are trying new approaches like the "flipped classroom" [14], distance learning [15], enquiry-based learning [16], and "do-it-yourself" (DIY) learning [17] that significantly modifies the dynamics and purpose of time spent in a formal learning environment.

In addition, several universities are beginning to offer courses online that have attracted tens of thousands of students per offering [18]. This educational model requires highly scalable tools to support content delivery as well as to evaluate the level of comprehension of both individuals and the class as a whole.

System & User Experience

There are a number of projects and products that provide similar feature sets and are being used at scale. Many of them provide practice problems to users, some personalize their content based on historical user performance, and some others use an interface similar to the one used by REGIS. Most of the well-known tools provide a curated bank of questions and do not allow others (instructors, for example) to organize or contribute questions.

Services

Project Euler [19] is a website that provides a large ($>$ 375) database of mathematical problems in a list-based interface. Finding solutions requires the use of computation and solutions (text, not code) can be submitted and verified. After submitting a correct answer, users get access to a forum where different solutions can be discussed. Problems can only be added to the list by the site's administrators.

Coding Bat [20] provides a set of programming problems in a list-based interface where code is actually submitted to the web server for execution and evaluation. Java and Python are both supported. A number of test cases are then run on the code to check for correctness. Again, questions can only be added by the site's administrators.

Google's *Code Jam* [21] competition uses an online tool for submitting solutions for review. Text-based solutions, not code, are submitted to the Code Jam servers. These questions are presented individually through a series of competitive rounds and again can only be added by the site's administrators.

Memorize.com [22] provides a digital flash card service where users can create simple flash cards for memorization-based tasks. Several assessment modes (flashcard, matching, and multiple choice) are available for testing knowledge, but all information must be entered manually. Pages can be shared with others after they've been created.

Academic prototypes or projects
Course Sharing [23] is a project in development at UC Berkeley that can host curricula for online courses including videos, lecture materials, and a variety of different types of questions. Questions can be customized for each student. This project is still very young but is designed to be administered by the course creator (a professor, for example) and distributed to students and does not allow students to contribute their own questions at this point.

edX [24] is a joint effort from MIT and Harvard that recently announced and is functionally similar to Course Sharing. The platform supports discussion groups, lecture content, wiki-based tools, assessments, and online labs. This project is also very new but aims to host entire courses (similar to Course Sharing) that are delivered from instructors to students, which varies significantly from REGIS's question-centric and democratized content creation approach.

Commercial products
Khan Academy [25] is a well-known non-profit company based around using online video for communicating knowledge and has designed an assessment platform that personalizes content for each user. Instructors can contribute their own problem sets and solvers using Javascript. This platform has improved significantly in the past year and has added features that bring it close to our tool's offering in many ways, although it lacks the functionality to allow students to create questions to share with their peers.

Programr.com [26] is a company that has designed a series of tutorials and programming challenges that can be used to practice programming skills. Like many of the other tools, the problems are defined by the site administrators and are not customized for each user.

MasteringPhysics [27] is a product from Pearson that provides simulation-based tools for a number of academic subjects. Instructors can add their own problems, but the problems are not customized for each student. Student performance analytics are also readily available for instructors.

Turing's Craft [28] is an interactive programming product that provides personalized feedback to introductory-level programmers working in Python, Java, C, C++ and several other languages. Instructors can now add their own exercises, but students cannot. The exercises are presented as a series of challenges and each challenge must be completed in order for the student to move on to the next exercise.

Curriculum & Activity Personalization

The order that a learner receives information can significantly impact their ability to comprehend the material being presented to them; in fact, information sequencing is one of the more important skills of an effective educator [29]. Even in the case of reviewing known information, certain problems will likely be easier to decompose than others; in computer science, for example, two problems that are phrased differently can vary significantly in difficulty, even if the underlying algorithm is the same.

Item Response Theory (IRT) [30] is one of the most widely recognized assessment-oriented approaches for ordering content delivery and provides mechanisms for evaluating the effectiveness of particular assessment items as well as the assessment as a whole. IRT methods evaluate the "ability" of an individual against several parameters of the assessment item, including difficulty, separability, and the likelihood of guessing the item by chance. REGIS uses a variation of the standard unidimensional IRT assessment function to perform its sequencing.

MECHANICS AND MAJOR DESIGN DECISIONS

Core Goals

There are three primary goals that have guided our design as the project has developed.

Goal #1: Simple and straightforward interactions
A number of digital alternatives exist for distributing practice questions to others, from general solutions like text documents and email to domain-specific tools like Project Euler and Coding Bat. These tools, while often simple, are not

optimized for the dynamic and inherently social nature of learning and don't allow for content customization and personalization.

One of the primary objectives of REGIS is to provide a tool that makes collaboration and review both easy and effective. No tutorial should be necessary to use REGIS effectively, and interactions should be self-descriptive whenever possible. The goal of the user is to create and answer questions, and the user interface aims to support those goals before all others.

Goal #2: Engaging and effective motivational tool
The ideal tool for our course would improve student understanding and interest in computing topics without adding significantly to student workload. Students could push themselves as hard as they wanted to, and students at all levels could find useful resources to improve their understanding.

Developing both cognitive aptitude and interest simultaneously requires both an effective tool and a selection of appropriate challenges. REGIS aims to provide a very clear set of interactions that are intuitive and non-intrusive; in addition, it makes it very easy for our teaching team to add new questions and determine which questions are being answered frequently as a proxy for measuring quality and interest.

Goal #3: Customizable and personalizable per-user experience
Personalized web-based learning has been the subject of a significant body of research, but tools that are both affordable and flexible to make this a reality are in relatively short supply.

REGIS provides a free solution that can be deployed in a variety of contexts and subject areas. The tool can generate a recommended question ordering for each user based on their historical performance and allows both students and instructors to create their own questions and share them with others.

What REGIS is Not

It is important to bound the expectation of scope for REGIS in order to disambiguate its purpose from preexisting tools.

REGIS is a tool for developing and distributing customized problems to groups of people. We have designed it primarily as a tool to supplement other mechanisms (classroom time, textbooks, online videos, testing systems, etc), not to replace them or duplicate their functionality. REGIS, therefore, is not:

A learning management system (LMS) or course management system (CMS). It would be more appropriate to think of REGIS as a possible subcomponent or extension to other LMS / CMS systems (Blackboard [31], Sakai [32], Moodle [33]); REGIS was designed to be able to connect to question providers that could potentially be run with any of these systems, but does not aim to replicate the majority of the functionality of a traditional LMS / CMS.

A Q&A tool. Several tools, most notably Piazza [34], exist that are designed for general question-and-answer exchanges between individuals ("when and where are the final exam for this course?"). REGIS is not designed to handle this type of question and would be a relatively poor choice for this level of functionality since users are given separate instances of questions and answers from others are not immediately revealed, even after they have been submitted.

Primary Design Decisions

Throughout the development process we have been continuously striving to keep intended user interactions both simple and self-descriptive in order to make the tool widely accessible and easy to learn. A number of important design decisions have been made in order to make the tool both easy to use and powerful, the most significant of which are defined below.

Language agnosticism. A number of tools are now available that allow learners to write and then submit or execute code directly in the browser[1]. Tools currently exist for common languages like Javascript, Java, and Python. While these tools have received a significant amount of well-deserved publicity, they each only support a subset of all popular languages and require considerable attention be paid to the security model when the user's submission is executed on the server in order to properly account for the risks of arbitrary code execution. Insufficient preparation can have significant negative side effects if malicious users choose to attack the system.

A far simpler approach from an administrative standpoint is to allow users to submit their answers, not the code used to generate the answer. This approach is also taken by Project Euler and Google's Code Jam.

In addition to the significant gains in system-level simplicity, the introductory computing course that REGIS was designed for uses a language [35] that currently has no automatic execution mechanism (such as a command-line interpreter). Programs therefore cannot be automatically executed and evaluated on the server-side unless such a tool is first built, which is a significant undertaking of its own.

[1] CodeAcademy [47], CodingBat, and Turing's Craft, for example.

Language agnosticism also makes REGIS immediately usable in any context, including those that have nothing to do with programming. The entire category of free response questions as well as knowledge domains like history and art, for example, are usually not affected by the system's dependence on a particular programming language. This was important for our goal of expanding the usefulness and applicability of the tool.

Simple UI and all interactions based on well-known flash card analogy. REGIS has a simple user interface that is entirely focused on questions and their solutions. We have iterated several times to reduce the number of distracting UI elements and make the tool more question-oriented. We have also had a goal of making the UI modern, fast, and fun to use without distracting the user.

In addition, it is possible for a user to solve the same question any number of times. Like traditional flash cards, it can be helpful to be able to go over questions that you've successfully answered in the past in order to review at a later time. This is the default behavior in REGIS.

"League"-based hierarchical organization with privacy settings. Questions are shared among members of a "league," which can be used to represent a single event, course, academic topic, or other group of users. A user can be a member of any number of leagues at the same time. Leagues are very easy to create and destroy for individuals with appropriate permissions and questions can be duplicated across leagues if desired. This makes it possible to create new leagues that inherit a set of questions but function independently.

Our course uses a single league for all students. Students could, however, be enrolled in other leagues simultaneously that represent other courses or even informal groups of students who are generating their own problems. Leagues can also be used for different sections of the same class if a distinct set of questions is desired for each section.

Primarily non-competitive. REGIS intentionally avoids framing questions competitively. Although competitions can be effective for motivating participation, they can also be discouraging (particularly for those new to the topic, such as the students in our introductory course) and can underemphasize collaboration. Our course aims to be highly inclusive and strongly encourages collaboration; the core REGIS system therefore does not use competitive mechanics like leaderboards, time trials / races, or explicit ranking aside from a small number of anonymous statistical comparisons to other students. Future work could include competitive modules that allow students to build teams and challenge each other to competitions.

Cognitive noise reduction. Questions are only answerable when there is a single question visible on the screen; it is possible to view multiple questions at a time for navigational purposes but the user must focus on a single question before being allowed to answer it.

In addition, it is possible to release questions slowly in order to reduce the perceived burden of having "too much left to do;" instead of overwhelming students with a large number of questions, some of which may be too difficult for particular students, REGIS can be configured to release targeted questions at a configurable frequency (or all at once).

Question groups, or "decks." Question groups, called "decks," allow users to organize questions into named collections. Questions can exist in multiple decks simultaneously and users can create their own decks and decide to keep them private or share them with others.

We've tried to keep the design of decks very general to cover a wide range of use cases, keeping exam review sessions, study groups, and exam administration as primary models.

Exam mode. Exam mode allows instructors to hide all questions except for those present in a particular deck. We added this feature based on feedback collected during informal surveys administered to high school teachers; the goal is to include all necessary features required to make it easy to administer tests at scale. The teachers who were surveyed believed that REGIS could provide a natural framework for administering tests if certain parameters could be tweaked when exams were in session.

Exam mode can be toggled on a per-league basis. Entering exam mode currently modifies four operational mechanics for the target league:

- Anonymous members cannot answer questions and each logged-in member can only provide a single submission per question.
- All questions in the exam deck become visible to all users in the league who can see the deck.
- All non-exam decks are hidden for the duration of the exam.
- New users cannot join the league while it is in exam mode.

Custom fields in shared question templates. Questions can be static or dynamic; dynamic questions include fields that can be customized on a per-user basis. The core of the question usually stays the same in dynamic questions, although the particular problem that the user is trying to solve will be different than those of their peers; for example, the specific values that appear in a word problem may be different for each users but the body of the word problem would be the same.

The benefits of the template-based question approach are twofold: first, we hypothesize that the model will promote appropriate collaboration by encouraging students to discuss approaches and algorithms instead of exchanging answers. Secondly, the modification of inputs makes it significantly more difficult for users to copy answers from their peers (during assessments, for example) and provides an easy way to generate an arbitrary number of different version of a particular exercise.

The language used to customize question variables was designed to be very simple and extensible in order to make question writing possible for non-programmers. Question variables, or "terms," are based on an extensible pre-built function library and can be included in the question with a minimal amount of syntax. This makes it relatively easy for non-Python programmers to write questions quickly without having to program, and also makes it very quick for those comfortable with programming to produce questions using the shared library. Some examples of terms that are currently implemented include:

`num` *`min max`* - generates a random number between min and max.

`mixset` *`filename (set-length)`* - this term operates on a particular file that should be a collection of values, one per line, partitioned into sets. A set is divided by placing an empty line between it and another set. `mixset` selects a random set (one of the appropriate size when provided) and scrambles all of the elements in the set randomly.

`link` *`data`* - this term produces a URL that links to a text file and drops data in as the text body. This will produce a link titled "View link" in the question body. Each question can currently have at most one link in its body due to the way that URLs to the linked files are generated.

A list of all available terms, including usage examples, is included in the appendix; new terms can be added easily by building a Python class that produces the desired output given the parameters. The term definition framework has been designed to be highly modular so that they can be shared easily among REGIS deployments. A sample dynamic question template that made use of these terms could be written like this:

```
How many Friday the 13th's will occur between Jan 1, 2011 and
[date_str]?

; day: num 1 28
; month: num 1 12
; year: num 2015 2020
; date_str: formatdate month day year
```

`day`, `month`, `year`, and `date_str` are all question variables, although `date_str` is the only one that actually appears in the question body. `num` and `formatdate` are both terms, and the parameters after each are arguments for the term. Based on the current term definitions, one possible permutation of this question might be:

```
How many Friday the 13th's will occur between Jan 1, 2011 and May
24, 2018?
```

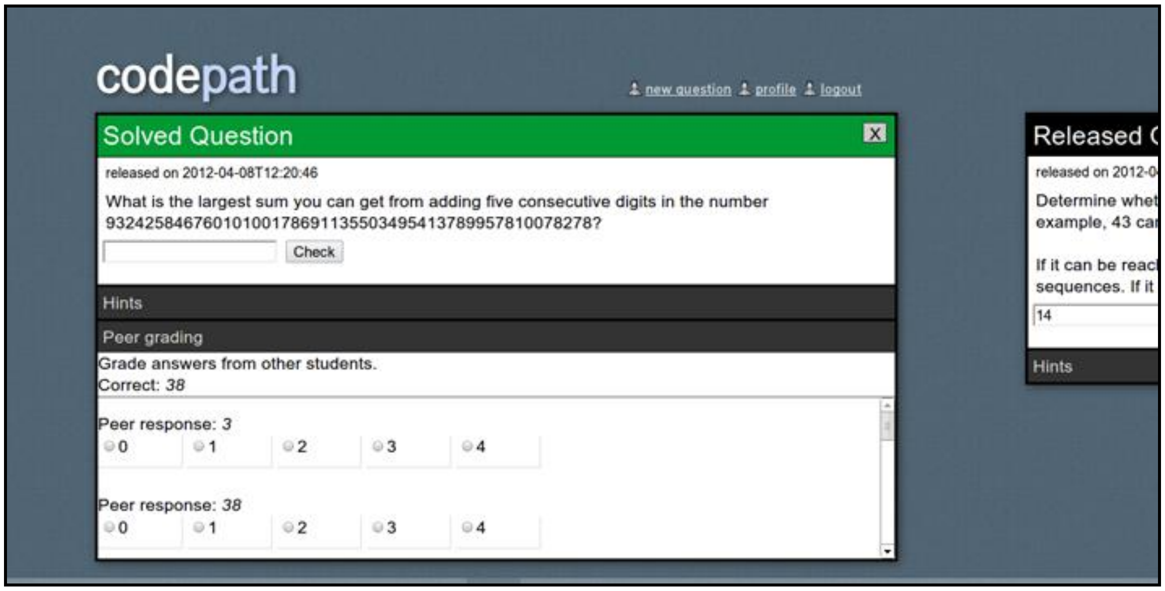
In this example, `day = 24`, `month = 5`, `year = 2018`, and `date_str = "May 24, 2018"`.

Democratic question submission. Questions can be submitted by any user in a league. Doing so will automatically generate new instances of the question (see next section) and bind an instance to each user in the league. Users should see the question appear in their "All" deck shortly after the question is created.

Users can create customizable fields in the questions that they submit. However, due to security concerns, questions may not be automatically graded because doing so would require the execution of a solver script. Since this script cannot be blindly trusted from all users, user-submitted questions must be manually graded (see below). It is possible, of course, for instructors to automate user questions by providing the solver script themselves.

In order to prevent undesired behavior (spam, low quality questions), a user's name is displayed with each question that they submit. Particular questions can also be flagged; if enough of a particular user's questions are flagged then their ability to create questions will be suspended.

Peer grading. In addition to being automatically graded, answers to questions can also be reviewed and scored by peers after the peer has answered the question themselves. This is presented as a simple UI component under the question itself, and users can grade and number of responses that they want to. The grading task is presented as a five-point Likert scale widget with brief descriptions for each point. The solution is also displayed to simplify the grading task and reduce errors.



released on 2012-04-08T12:20:46

What is the largest sum you can get from adding five consecutive digits in the number 93242584676010100178691135503495413789957810078278?

Grade answers from other students.
Correct: *38*

Peer response: *3*
0 1 2 3 4

Peer response: *38*
0 1 2 3 4

released on 2012-0

Determine whet
example, 43 ca

If it can be reac
sequences. If it

14

Figure 3: Peer responses appear below a question after a user has answered it correctly. The correct answer is also displayed to simplify the evaluation task.

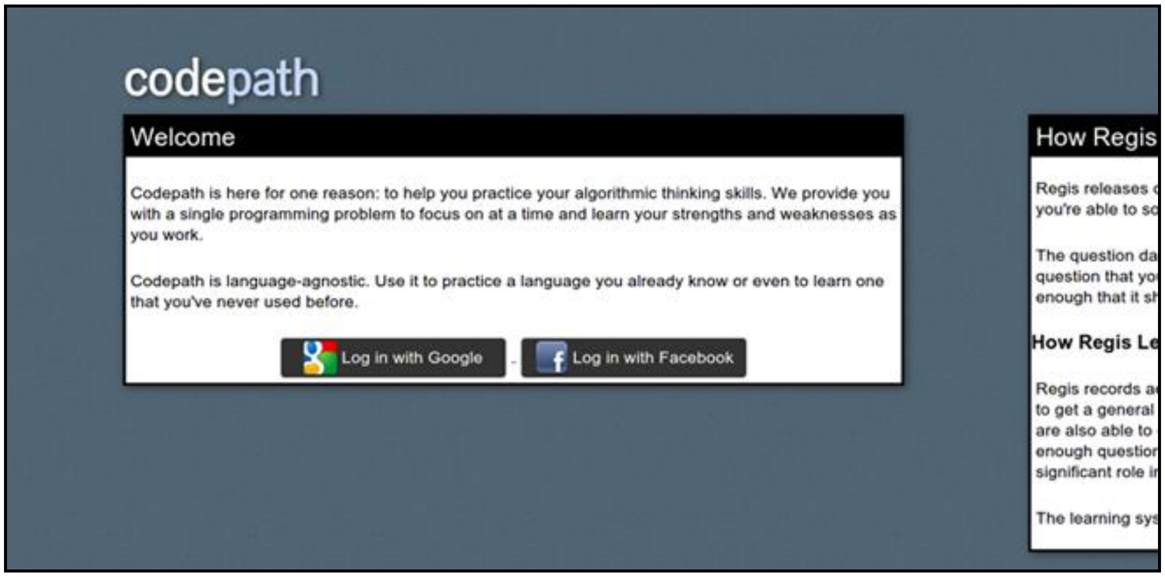
This was included as a method for supporting both democratic question submission and additional question types that are challenging to automatically evaluate (free response, for example). It also provides a means for instructors to crowdsource the distribution of feedback and provides an opportunity for students to see how their peers respond to questions that they've already answered. Peer grading has been evaluated in similar contexts recently and found to be relatively dependable [36].

Frontend: User Interface & Experience

The REGIS frontend is modeled after the analogy of flash cards used in a number of academic environments but most commonly for studying and reviewing content. Flash cards are an incredibly simple tool and REGIS attempts to emulate the simplicity of flash cards while also improving on the standard physical objects, which are limited in scope by their static nature.

When users first access the site, they are asked to log in using either their Facebook or Google account[2]. After a user logs in, all further interactions are handled using asynchronous calls to the server using the Javascript frontend, meaning that no full page refreshes occur and very little pausing takes place in the interface as a whole.

[2] REGIS also supports local authentication but it is disabled by default.


codepath

Welcome

Codepath is here for one reason: to help you practice your algorithmic thinking skills. We provide you with a single programming problem to focus on at a time and learn your strengths and weaknesses as you work.

Codepath is language-agnostic. Use it to practice a language you already know or even to learn one that you've never used before.

How Regis

Regis releases

you're able to s

The question da

question that yo

enough that it s

How Regis Le

Regis records a

to get a general

are also able to

enough questio

significant role i

The learning sy

Figure 4: The landing page uses a layout that is very similar to the project's core user interface. Users have the option of logging in to the service using either Google or Facebook credentials, or they can flip between some of the other introductory cards to learn about how the tool works.

Once a user logs in they are presented with their profile card, which contains user statistics and links to questions, in the center of the user interface. Each card displays the question text and provides a place for the user to submit their answer, as shown in Figure 1. In addition, cards can display a number of possible "scraps" (described below) to supplement the question content.

The user can then switch to other cards in the current deck using either the mouse (by clicking on buttons on either side of the active card) or the keyboard (using the left and right arrow keys). This allows users to move linearly through the deck. In order to view all questions in the deck at once, users can "stack" all of the cards (again, using the mouse or keyboard) in a column and select the question that they want to focus on.

Decks

A deck is a simply a user-specified collection of questions. Users can create and name their own decks and add any number of questions to each one. Decks can be private or shared with the league. A single question can exist in any number of decks; the REGIS server actually stores the relationships between decks and question templates, not the question instances that users actually interact with. This means that users will still receive their permutation of a particular question if someone else adds the question to a public deck.

Decks have been designed as a general organizational mechanism that can be used in a variety of settings. Some possible use cases include:

Purpose-driven decks, such as creating decks of review questions. These can include questions from the user, other members of the league, and the instructors.

Progress-driven decks that keep track of known concepts and unknown concepts (similar to how standard flashcards are often used).

Organizational decks that group questions into topics or into visibility categories.

League instructors can also launch *exam mode* by selecting a deck or decks that should be visible during an exam. These decks then become visible to the league and all other decks are hidden for the duration of the exam.

The lower portion of the REGIS user interface contains a collapsible "deck dock" (see Figure 1) that provides a simple way for users to interact with their available decks. Clicking on a deck or pressing the corresponding number key (1 for the first deck, 2 for the second deck, etc) activates the deck and refreshes the visible card list to contain only questions that exist in that deck. Users can also create new decks, drag cards onto decks to add them to the deck, and "close" cards in order to remove them from the current deck.

Scraps

Scraps are small UI elements that appear below questions in order to share specific information about the question and offer additional interactions. Scraps are intended to be extensible and pluggable so that different deployments of REGIS can use different scraps.

Scraps appear as small cards that are connected to the primary question card that they refer to (see Figure 3). Two types of scraps that have currently been implemented are hint scraps and feedback scraps.

A hint scrap is used to either (a) request a new hint, (b) display a hint that's been requested or (c) leave a hint after successfully answering the question. If the question is unanswered then the user can see the hints they've requested and request new ones (if available). If the question has already been answered correctly then the user has an opportunity to leave a hint for others. Hints also receive feedback scores by users who view them, and these scores probabilistically influence the likelihood that the hint will show up when a user requests a hint.

A feedback scrap is used to collect feedback about the user's opinions on the question, both in terms of general quality and difficulty. Both are presented with a five-point Likert scale, and leaving feedback is optional for all questions. This data is currently only being recorded, although it would be possible to incorporate it into REGIS in the future.

ARCHITECTURE OVERVIEW

REGIS has been designed to be a modular system whenever possible in order to support a variety of different use cases. The system architecture is similar in many ways to those of modern web applications and can easily be deployed on a single system or multiple systems.

At a high level, REGIS is structured as depicted below.

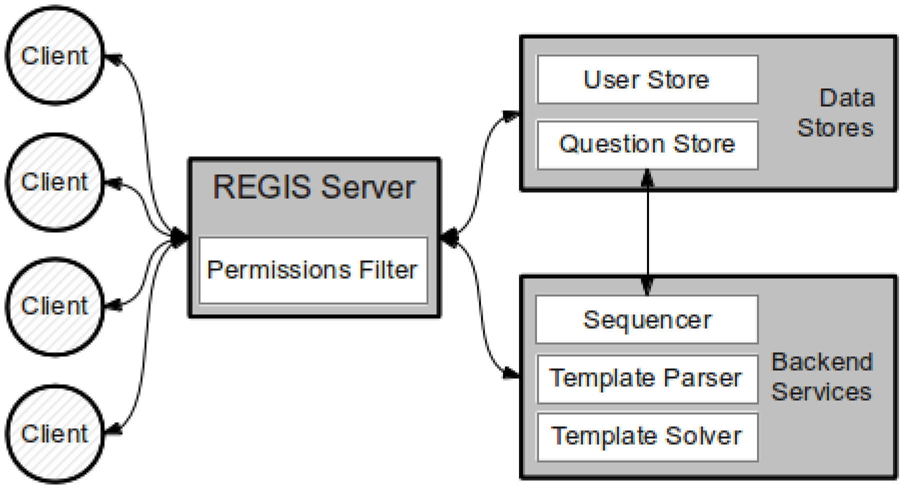


Figure 5: the REGIS server is relatively lightweight and can support a large number of concurrent users. The majority of the processing work is offloaded to the backend services, which can be run at off-peak times or on separate machines if needed.

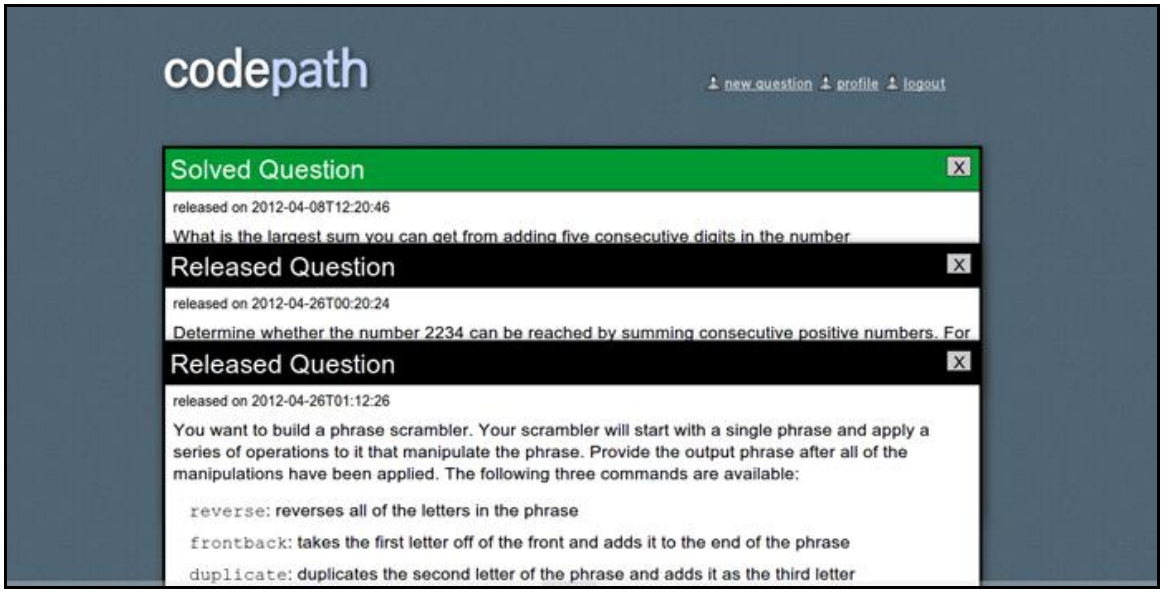
It is possible for all server-side processes to take place on the same physical machine unless a very high level of participation is achieved or new question instances are being generated with high frequency. The majority of server-side operations are very lightweight with the exception of the backend services that handle the solving of questions. Each of the permutations of a question must be solved when the question is first introduced into the system; if a large number of permutations for a computationally challenging question are required then CPU utilization will peak until the answers have all been generated. In this relatively unusual case (such as with high-enrollment online courses), the computationally-intensive portions of REGIS can easily be run at off-peak times or shifted to other nodes if required.

Client Frontend

The frontend is presented to users in a web browser and has been built using two popular Javascript libraries, jQuery [37] and Backbone.js [38]. The client makes a request to pull all necessary information from the REGIS server when the user logs in -- this includes all available questions, decks, and user profile information. The majority of a user's question-related data, however, is stored locally in the client, which allows for fast interface response times and no full-page refreshes to transition between client-side actions.

Solutions are not sent to the client in order to prevent the users from reading them before the problem has been solved; checking answers, as well as submitting new questions and modifying deck memberships, require communication with the server backend. The client sends asynchronous requests as needed in order to keep client and server state in sync. For example, when users add a card to a deck or create a new deck, an asynchronous message is sent to the REGIS server in order to make the change persistent across sessions.

The frontend has been through several design iterations, but is now relatively simple and is based around the metaphor of flash cards. Users are able to interact with the cards in natural ways, including dragging them into decks to add them to a deck, dragging decks onto the stage to reveal all questions in the deck, stacking cards to view many at the same time, and so on.



released on 2012-04-08T12:20:46

What is the largest sum you can get from adding five consecutive digits in the number

released on 2012-04-26T00:20:24

Determine whether the number 2234 can be reached by summing consecutive positive numbers. For

released on 2012-04-26T01:12:26

You want to build a phrase scrambler. Your scrambler will start with a single phrase and apply a series of operations to it that manipulate the phrase. Provide the output phrase after all of the manipulations have been applied. The following three commands are available:

`reverse`: reverses all of the letters in the phrase

`frontback`: takes the first letter off of the front and adds it to the end of the phrase

`duplicate`: duplicates the second letter of the phrase and adds it as the third letter

Figure 6: Cards can be stacked in order to simplify navigation. The deck dock has been collapsed in this view.

The user interface is divided into two primary regions: the stage and the deck dock, as shown in Figure 1. The stage is the primary section of the page that contains all of the cards in the active deck. The deck dock occupies the lower portion of the interface and gives the user a simple method of switching between decks, adding cards to decks, and creating new decks. The deck dock can be collapsed when not in use.

REGIS Server

The REGIS server is primarily a content filtering and annotation pipeline that exists between the question provider and the client-side user interface. The server is responsible for responding to all client-side web requests, including requests for questions and decks. In order to answer question requests, the server forwards the request on to the question provider, which can either be stored locally or remotely using a service like Course Sharing. The server is written using the Django (Python) web framework and its responsibilities are limited to generating responses to RESTful client-side HTTP requests.

The REGIS server keeps track of viewing and editing user permissions for questions or decks and filters the content that is returned appropriately. This separation of responsibilities between the question provider and the server itself makes it possible for REGIS to apply a wide range of rules about which information is available to users at different times, such as a question-per-day release policy.

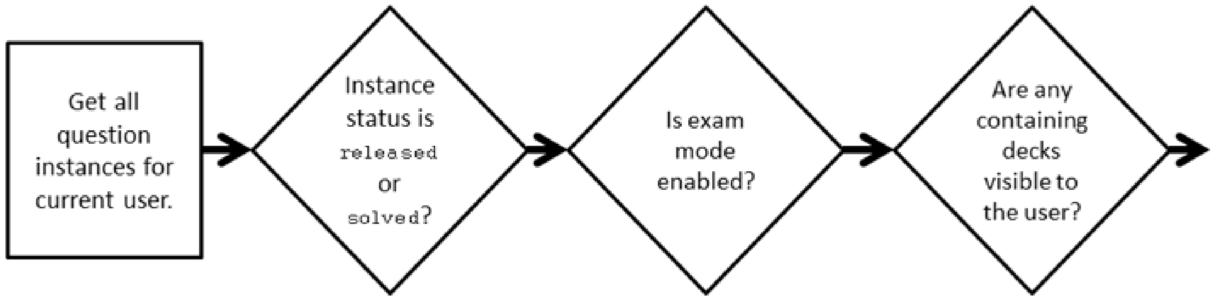

Get all question instances for current user.
Instance status is released or solved?
Is exam mode enabled?
Are any containing decks visible to the user?

Figure 7: Before a question is displayed, the REGIS server passes it through a visibility pipeline to determine whether the user should see it. The question is not displayed if it fails any of these conditions.

The server maintains all deck-related information, including which question templates are present in which decks, deck visibility, and ownership information.

In addition, the server also maintains information about annotations to questions (hints, grades, comments, etc) that may not fit into a particular question provider's schema. This information can be added to each question as it is requested from the client, again, depending on viewing permissions.

One notable task that the REGIS server is not responsible for is user authentication. This responsibility is currently offloaded onto popular identity providers like Google and Facebook through the use of their OAuth API's in order to lower the barrier to participation. A user table is still maintained locally to allow accounts to persist between sessions but no passwords or other sensitive data (except for the user's email address and name) are stored in the system. Django has an authentication system built-in and local accounts could be easily implemented if desired for a particular application.

Question & User Stores

Questions are the core unit of data that is handled by REGIS and the system is entirely capable of storing all question-related information locally. However, it's not clear that this will be the best solution in all situations, especially due to the recent proliferation of education-related technologies like learning management systems and online quiz platforms. Because of this, REGIS makes it very simple to connect with other third-party backends to fetch question data and store guess data through the use of question provider modules. Two modules currently exist: the LocalQuestionProvider and the CourseSharingQuestionProvider.

The local provider stores all information on the same system as the REGIS server using the same database backend that is used for the Django deployment. This results in relatively high performance since no remote queries take place, but the database may suffer some performance degradation if query volume reaches levels beyond what the server can handle in addition to its filtering and annotation tasks. This circumstance should be very unusual unless a large number of users tried to sign in simultaneously, such as for a test in a high-enrollment online course.

The CourseSharing provider stores and retrieves questions and answers using the online CourseSharing service being developed at UC Berkeley. Using this provider allows questions to be shared among other CourseSharing services and for user performance to be aggregated among those services as well.

Backend Services

There are three primary backend services that keep REGIS functioning properly: the question parsing service (required unless a remote question provider with its own generation mechanism is being used), the question solving service, and the sequencing service. All of these can be run periodically and must have access to the REGIS question store. A technical overview of all three services will be provided in a later section.

The question parsing service creates new question instances that are assigned to users as their accounts are created. Question instances are generated according to the variable definitions specified in a template, and a pool of questions is commonly generated for each template. A question instance is coupled with the set of all significant answers[3] (assuming that the question is marked as computable; the alternative is peer-reviewed).

The personalization service estimates the difficulty of each question and the "skill level" of each user by retrieving aggregate performance data. The order that questions will be released is then adjusted for each user; this field is used to determine what question is released when needed. The ranking that a question receives is based on the proximity of a user's skill (both globally and for the categories that the particular question falls in) and the question's difficulty (predicted based on attempts from other users and the categories that the question is classified in by the question author). The precise formulas for how the rankings are determined are described in the *Questions* section below.

Question Generation

REGIS has a simple and extensible question generation engine built in that makes it relatively easy to write new questions. The parser process is responsible for generating new question instances from question templates and storing this information with the question provider. The solver process determines what the correct (and in some cases, incorrect) answers are for each parsed question and stores each with the question provider. In the case of the LocalQuestionProvider, each question instance is stored as a row in the questions table and each answer in a separate record in the answers table, which is linked to the question that it answers, and inlcudes a flag indicating whether it represents a correct answer.

It is important to note that the components described in this section are only used if the question generation tools that come with REGIS are used; if another method for generating questions and answers is preferred, then the question provider can be updated using those mechanisms instead with no further changes to the REGIS infrastructure.

Question Data Model

The question data model was designed to be easy to customize, easy to share, and easy to scale. Question templates consist of two components: the template text (required) and the solver script (optional). The authors must provide a solver script if the question is going to be evaluated automatically. These two components can be shared among REGIS distributions in order to share particular questions.

[3] A significant answer can be either correct or incorrect but is generated by a specific solver in order to either (a) indicate a correct response or (b) provide customized feedback for an incorrect response.

The template contains the body of each question and any number of variables that can be generated on a per-instance basis; question instances are generated from templates by the parser. The solver script, if available, is executed by the solver and will generate unique entries for answers based on the values of the variables chosen by the parser, which are stored with the question instance in the question provider and passed as inputs to the solver script. Questions can have any number of correct answers as long as the solver script defines how to compute them. Questions can also have any number of incorrect answers that receive customized responses; these are defined in a separate method in the same solver.

Templates can also be tagged with certain "categories," which are used by the sequencing algorithm to identify questions that are topically similar. These categories are not currently exposed in the user interface and are used solely for personalization services.

Each question has a status associated with it that gets updated as it moves through the pipeline. Users do not see a question until it is marked as `released`, meaning that both the question text and the solution(s), if applicable, have been generated and the question has been released to the intended user. A question has one of the following statuses at all times:

`pending` - hidden; parsed but not solution not yet computed.
`ready` - hidden but parsed with a computed solution. This status is used when questions are ready to be revealed but the league's configuration restricts their visibility (i.e. if the league operates under a periodic release schedule); if the league's policy states that questions are automatically released then this state will not be used.
`released` - visible and answerable by the user
`solved` - visible but not answerable by the user
`retired` - was once visible but is no longer (should be considered "deleted" from the user's perspective)

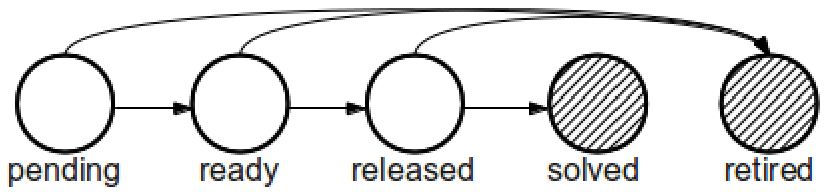


Figure 8: Questions move through a series of states that determine the visibility and answerability of the question.

It is inevitable that question submitters will eventually need to be able to correct the wording or variable definitions in their questions after the question has already been revealed to users. Direct deletion of questions is never performed in order to maintain the validity of the guess history; instead, the question can be *refreshed*, which converts all old versions of questions to `retired` unless the question has already been solved, in which case the question is left alone. A new version of each question is generated for each instance that was retired using the new template, and certain metadata is copied over as well (the most significant being the release date of the original question for the target user) in order to make the refresh operation transparent to the user.

The system's original question model generated a set of new question permutations for each user. This made it possible to increase the variation between users by generating random outputs for each person, but came at the cost of a question database whose size depended both on the number of templates and the numbers of users. This was not problematic for our test deployment but could quickly become prohibitive if launched as a public project (like Project Euler) or used for a large online class (like Stanford's recent offerings, several of which had over one hundred thousand students enrolled).

REGIS's current question model generates a pool of questions for each template; the template pool size is customizable per deployment. Each user is paired with a random selection from each pool when their account is first created, and additional pairs are formed (in case new templates are added) each time a user logs in. Any number of users can be paired with a single instance of a parsed question template, meaning that the question database's size is dependent only on the number of templates. Templates that do not contain any variables only generate a single instance by default, since all instances would be equivalent.

A user is matched with an instance from each active template as soon as the user's account is created; this may generate unused edges between users and question instances (since many users may not see all questions), but provides records for the personalization services to manipulate. These records are required to guarantee that all questions are ordered correctly by the personalization service.

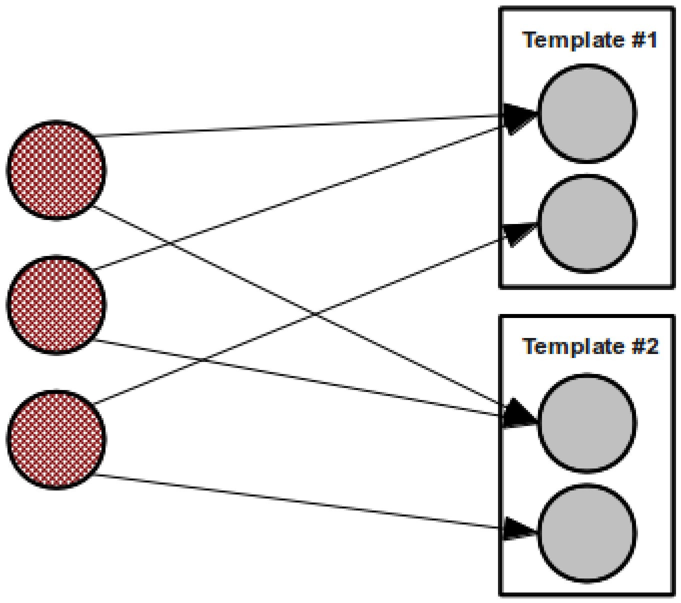


Figure 9: Multiple users can be matched with the same question instance; this architecture allows REGIS to maintain a bounded (and configurable) database size while also ensuring that users have a high probability of receiving different question instances.

Parser

The parser is run to generate new question instances based on a question template. It pulls templates from the REGIS database and generates a pool of instances for all active templates that do not have full pools. It also scans through any pre-existing question sets and replaces the questions that have been retired but do not yet have a successor.

The parser for REGIS is designed to be able to handle variable definitions but does not allow inline conditionals, function definitions, or other common language features. These were excluded in an effort to keep the required knowledge threshold as low as possible, and with the understanding that these operations can be included in the term's definition (which can be written using the full Python language) if required.

Variables can also be passed into terms as parameters; this cannot be done cyclically, however, as each variable must be defined before it is used as a parameter. Variables can otherwise be defined at any point in the question template.

The question template is written using the syntax described in *Mechanics and Major Design Decisions* section above; note that a particular template may also depend on the availability of certain terms, which must be transferred with the template if the template is shared between distributions.

As described earlier, *terms* are the functions that define how values for variables will be generated. The `num` term, for example, generates a number between the value specified by its first parameter and its second parameter (inclusive; `num 1 10` could generate any value between 1 and 10). Additional terms can be defined by implementing a single method in an abstract Python class (the base `Term` class). This method will receive all input parameters and can handle them in a native Python environment.

Solver

Solvers receive the inputs for a particular instance and generate a set of all significant answers. In addition to finding correct solutions, each solver can also compute any number of specific *incorrect* answers and provide customized feedback for the answer when a user enters it. For example, in a straightforward addition problem like:

```
[num1] + [num2]
```

If `num1 = 5` and `num2 = 2`, the solver could specify common error conditions as follows:

if `answer = 3`, say "check the operator...this isn't a subtraction problem"
if `answer = 10`, say "check the operator...this isn't a multiplication problem"

More advanced questions can include full algorithms with single steps left out. If the question has an intentionally tricky point in it then a hint could also be provided using this interface.

After a question has been solved, it transitions from the `pending` state to the `ready` state.

Questions

Questions are the core unit of interaction in REGIS. REGIS places questions into two primary categories, each of which is handled slightly differently:

1. *Automatic evaluation*: questions that can be graded automatically are evaluated online to provide users with immediate feedback. These questions require solver scripts.
2. *Manual evaluation*: questions that should be graded by individual or peer review.

REGIS is designed to allow any league member to submit manual evaluation questions but limits the creation of automatic evaluation questions to instructors due to the requirement of a solver script, which could contain malicious code and therefore opens a security threat on the system.

Question corpus

Our test deployment contained 24 instructor-provided questions that were pulled from a variety of sources or written by a member of our teaching staff. All of the questions could be automatically evaluated due to reasons described in the *Test Deployment* section below. The question templates were selected or designed such that each problem would be hard to solve without writing a computer program to do so, similar to the approach of Project Euler but with a lower expectation of math fluency. In addition, the problems were significantly more complex than traditional fill-in-the-blank style questions that appear on flash cards; all of them would require stepping away from the question, solving the problem for a targeted 5 - 120 minutes, and returning with the answer.

The particular questions we chose to include were selected because they involved interesting algorithmic questions that weren't immediately obvious from the way the question was worded. A significant subset of the questions required implementing algorithms that were more complex than those introduced in the course and required independent research in order to determine how to accurately solve the problem. These questions were included to explore whether students would be willing to perform mild to moderate amounts of investigation in order to determine how to solve a problem.

Some sample problems that were included in our initial launch set are included below.

> A McNugget number is a number that can be reached by adding some combination of 6, 9, and 20 (these are the different sizes of McNuggets that you can order at McDonalds) any number of times.
> How many of the following numbers are McNugget numbers?
> 176, 270, 349, 403, 501, 667, 794, 884, 914, 953
>
> Imagine that a ball is bouncing around a two dimensional grid and moving one step right and one step upward each second. When the ball hits a wall, the ball bounces off but maintains the same velocity. The direction, however, is reversed on the axis where it met the wall. If the grid is 41 x 41, the ball starts at (30, 28) and is moving northeast at one cell per second in each direction, what cell is the ball in after it two hundred seconds?
>
> Write the decimal number "18735" in binary.

How many ways can you make change in American currency for $0.74?

Determine whether the number 1782 can be reached by summing consecutive positive numbers. For example, 43 can be reached by summing 21 + 22, and 42 can be reached by summing 13 + 14 + 15. If it can be reached, provide the smallest number greater than zero that appears in one of these sequences. If it cannot be reached, provide the number 1782.

(credit to Project Euler for this question)

User-submitted questions

Any user in a league is allowed to submit questions to that league unless their privileges have been suspended for abuse. When a user submits a question, the question is stored as a template and is immediately parsed. The instances of each question are then lazily bound to users as they make their next question request.

Instructors have the ability to disable templates and the corresponding question instances as needed.

Question sequencing

A number of different approaches have been used for curriculum sequencing in online tools. REGIS currently uses a modified (and simplified) version of the technique described by Chen et al [39], which is itself a modification of the standard function from Item Response Theory [30].

Chen et al's work broadens the metric for sequencing to include measures that account for both estimations of the ability of the user and the difficulty of the material. The default sequencer in REGIS attempts to perform a similar operation by looking at implicit feedback received by the user's peers, similar to common collaborative filtering methods [40]. The sequencer accounts for four different measures for ordering:

User global ability

This measure estimates the user's general ability at answering questions in the corpus. This value can be used to compare users but is context-free, meaning that it does not weight questions differently based on their relative difficulty.

$$\mathrm{A}_{\mathrm{G}} = \frac{C_u * \theta_u}{V_u}$$

The variable C_Q = the number of questions that the current user has answered correctly, V_Q = the number of questions that are available to be answered by the current user, and θ is the persistence modifier (described below).

User local ability

A user's local ability adjusts the ability score based on the context of a particular question. The local ability score is based on a particular user's performance within a category of questions.

$$\mathrm{A_L} = \theta_u * \sum_c \frac{C_c}{V_c}$$

The value of C_C = the number of questions that the current user has answered from within the category *c*, θ_u is the persistence modifier, and V_C = the number of questions that are available to be answered by the current user from the category *c*.

Question global difficulty

This component estimates the difficulty of the question among all others. This quantity can be used to directly compare questions.

$$\mathrm{D_G} = \frac{C_Q * \theta_Q}{V_Q}$$

The value of C_u = the number of correct responses, V_u = the number of users who have access to the question, and θ_Q is the persistence modifier.

Question local difficulty

The local difficulty of a question is computed based on the categories that it is affiliated with.

$$\mathrm{D_L} = \theta_u * \sum_c \frac{C_{cu}}{V_c}$$

The value of C_{cu} = the number of correct responses that the user *u* has issued to questions in category *c* and V_c = the number of available questions available to all users in the category *c*.

These four measures can be computed very quickly, and the distance between each user's ability score and the question's difficulty score provides a method for ordering available questions. A small bias is also applied in favor of positive differences over negative ones in order to give a preference to questions that are slightly above the user's skill level instead of those slightly below it. The scores should then be negated by subtracting the score from the maximum possible score of 2.

Appropriateness of question for user u

Appropriateness is measured by finding the difference between the question's difficulty and the user's ability level. Difficulty and ability level are measured both holistically and modularly using question categories

$$\mathrm{Q_u} = \frac{2 - |D_G - A_G| + \frac{(|D_L - A_L|)}{n}}{2}$$

The value of A_G = the global ability score for the user u, A_{LC} = the local ability score for the user u on category c, n = the number of categories that the question is affiliated with.

Persistence modifier

The persistence modifier is intended to provide a confidence weight to each of the scores that estimates how well a particular user understands the questions that they answer correctly, or how well a particular question is understood by those who answer it. There are two similar equations used to compute the persistence modifier, depending on the score that's being computed:

$$\theta_u = \frac{C_u}{V_u + \sqrt{|V_u - X_u|}}$$

$$\theta_Q = \frac{C_Q}{V_Q + \sqrt{|V_Q - X_Q|}}$$

The values of the variables in these equations are as follows:

C_u = the number of correct responses that the user u has submitted
C_Q = the number of correct responses that have been submitted for question Q

V_u = the number of attempts that the user u has submitted
V_Q = the number of attempts that have been submitted for question Q

X_u = the average number of attempts submitted per user
X_Q = the average number of attempts submitted per question

When the output values of the above function are sorted in descending order, the scores provide a recommended ranking for questions for each user. REGIS currently interprets this as a probabilistic score; each question's final score is raised to a power β, initially 3, and then all scores are normalized and questions are selected randomly from the weighted range. This aims to prevent concept stagnation where recommendations are only given for topics that users have significant amounts of success with; instead, there is a higher probability that users will receive these questions but will likely receive improbable questions periodically, allowing the knowledge model to stay relatively fresh as the student continues to improve. REGIS currently computes this ordering for all unanswered questions and users once per day, but the frequency can be increased or decreased as required.

In addition, it may be possible to perform sequencing across leagues if correlations can be detected between a user's ability in different leagues. This has not been implemented or tested in REGIS but could be the subject of future work.

MODELS OF CLASSROOM INTEGRATION

REGIS can be integrated into classrooms in any number of ways. We have used REGIS for one semester and plan on using it in the future for both online and in-person contexts using a variety of different classroom strategies. We report on what we've done so far as well as some other strategies that we intend to try in the future.

Test Deployment

We deployed REGIS publicly at http://codepath.co and used the service in our introductory computing class starting on February 16, 2012. We initially started with a small set of 10 questions but gradually grew the deck to 24 questions as the semester progressed and we covered additional topics. The site was kept online continuously throughout the semester with the standard REGIS system rules, meaning that one new question unlocked if none were answered in 48 hours.

We decided to make involvement in Codepath (our deployment of REGIS) voluntary based on both the students' expressed desire to participate and the rapid (and occasionally unstable) pace of development throughout the semester. The existence of Codepath was announced after the students had their first major evaluation and several updates and reminders were posted periodically throughout the semester when new major features or questions were added. The system was occasionally used during office hours and review sessions for practicing.

We launched an early version of the project in our course and there was no support for user-submitted questions or peer grading. This functionality has now been added to the project but was not released to our students because the features weren't tested and ready until late in the semester. The flash card user interface was also not available in the version that we tested with students in our course.

Alternative Models

There are a variety of other methods that REGIS could be used in within the context of classrooms.

The first and perhaps most obvious of these is mandatory involvement; REGIS logs user actions and per-user statistics can be generated for users to incorporate into student evaluations. Students could, for example, receive a set of questions in a period of time and be required to solve one or more of the questions per period (each day or week, for example). Similarly, students could also be required to submit questions that would go into the pool for other students to answer. Homework assignments could be presented through the interface and automatically evaluated; tools like REGIS could also help confirm student understanding in student-led learning arrangements like the flipped classroom.

An alternative approach would involve using REGIS as an informal review tool, with questions generated by students, instructors, or both. Instructors could build decks for review sessions, or the sessions could take place entirely online. The tool could also be used simply for "review on-demand" since it is available at all times, customized for each student, and updated automatically when new questions are added or removed. This is comparable to the approach we took while implementing REGIS in our classroom environment.

RESULTS

We maintained an active REGIS deployment for fifty days during a recent semester. We launched the service approximately one month after the the course began (immediately after one exam) and report on usage results that cover a time period containing a single midterm exam.

Our REGIS deployment had a total of 178 unique users after fifty days of availability, including 99 who were identifiable as registered students in the course, 20 who were affiliated with the teaching team for the course, and 60 whose affiliation could not be identified by their registration information.

Initial engagement: account creation

After fifty days of availability, Codepath had 178 registered users including 99 that could be identified as students enrolled in the course. The remainder of the users were teaching assistants, local high school teachers, and a variety of users with unknown origin. Codepath's availability was not directly advertised beyond members of the course and occasional informal conversation with others (such as the high school teachers that we surveyed for early user feedback). Codepath did not rank highly in search results on common search engines.

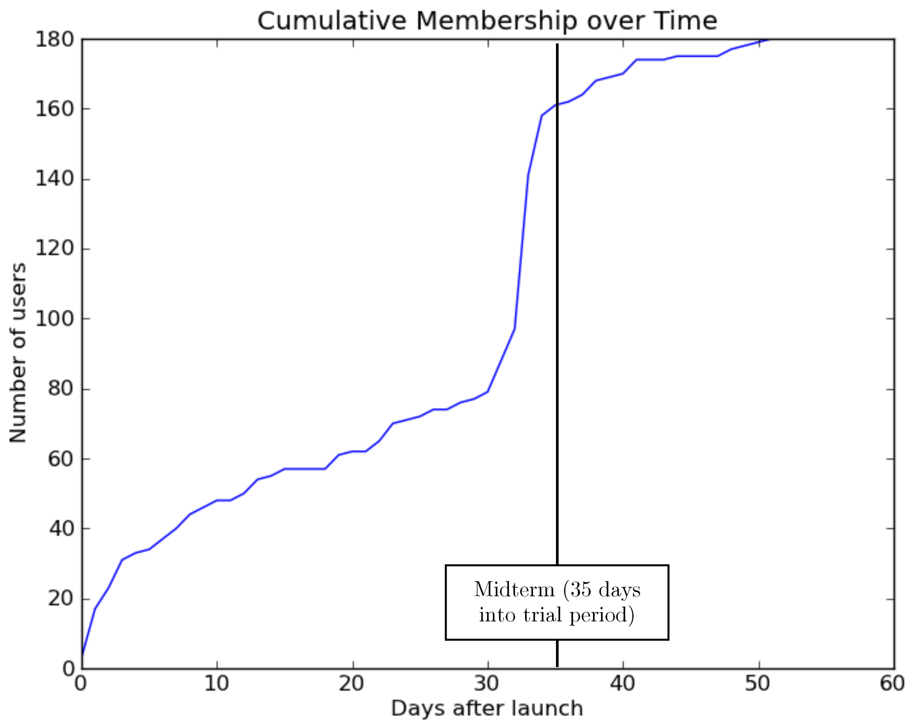


Figure 10: participation grew steadily for the first month after initial launch and spiked sharply around the time of our course's midterm exam.

Approximately 90.5% of users joined before the midterm, including 23.1% that joined in the first week of availability and 48.3% that joined during the week before the midterm.

Medium-term engagement: usage statistics

Despite the relatively high number of account creations, user involvement was relatively low when viewed over time. Many users (38.9%) did not log in again after they first created their account, and a significant portion of users (61.1%, including those who did not log in after initial account creation) never submitted a guess for a question. The top user, however, logged in 27 times and made 27 guesses. These usage patterns follow the same power law distribution for both engagement metrics that is detectable in almost every online interactive resource [41]. The "90/9/1 Principle" [42], states that the top 1% of users will be content "producers" who are highly involved, approximately 9% will be "editors" who contribute to the community in smaller ways, and 90% make very few contributions at all. A similar pattern has been found in a number of online communities and tools [43–45].

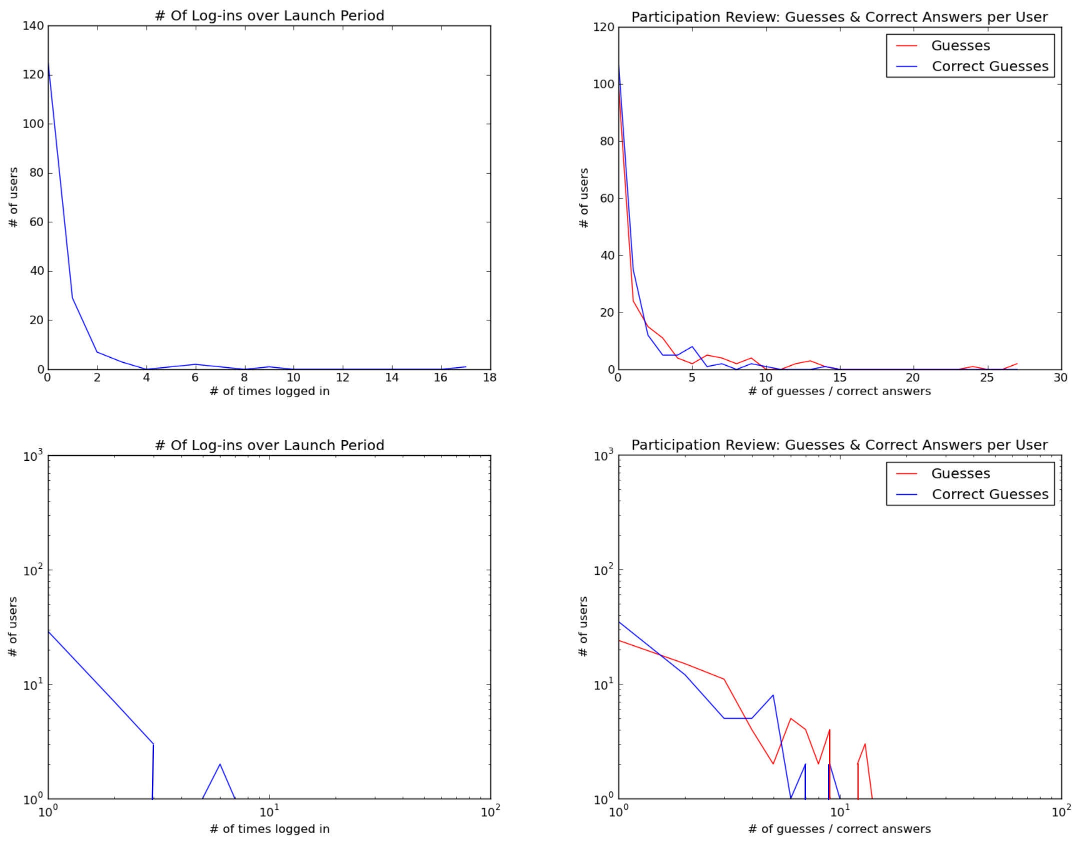


Figure 11: User login counts and guess counts follow similar power law distributions, as is often seen in online communities. Only a small number of users logged in more than five times and made more than ten guesses. The data is presented with linear axes on the top row and logarithmic axes on the bottom.

As an additional note, we suspect that a number of user logins were not registered by our logging system due to a bug that kept the event from being recorded if a user returned to the site after being previously logged in. The login events, therefore, only indicate occasions where users explicitly established a new session identifier and do not reflect multiple accesses from the same session.

Overall, the number of users who created accounts was somewhat higher than expected, with 42.1% of students in the course signing up for the service; active participation metrics, however, were lower than expected with a mean of .75 guesses per student (stddev = 2.95). The low participation rates are likely due to the fact that Codepath was presented as an optional exercise in a course that maintains a relatively high workload throughout the semester; in future semesters we will be testing alternative methods for integrating the tool into the course to test this hypothesis.

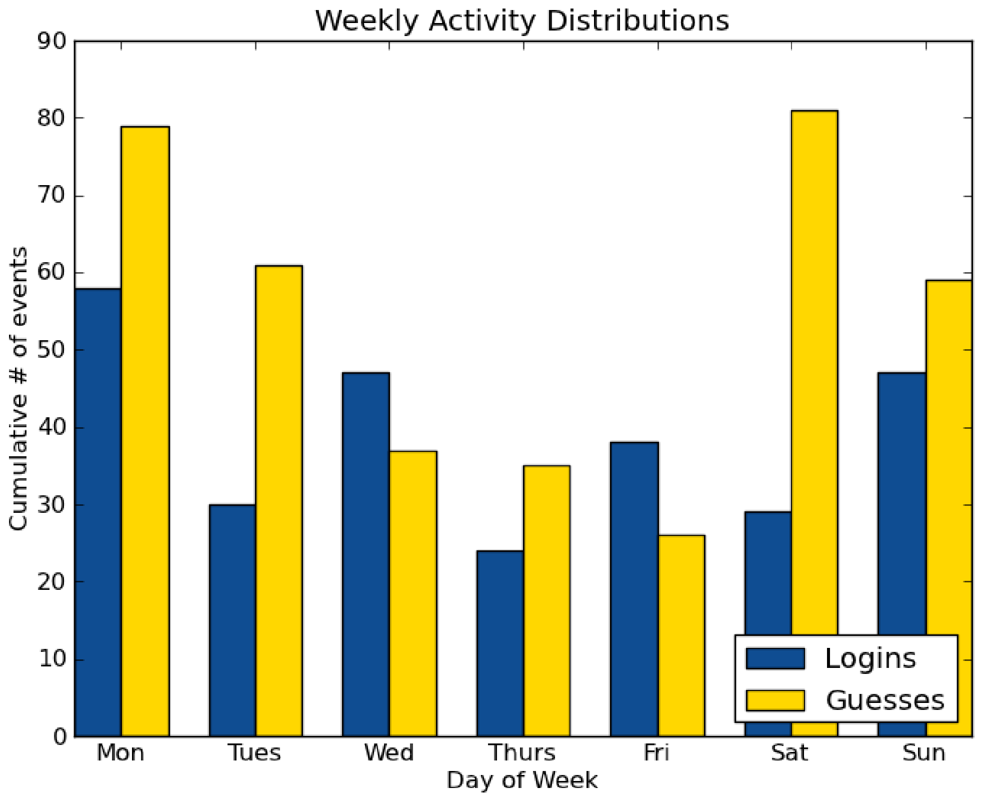


Figure 12: users logged in most on Mondays, Wednesdays and Sundays. The most popular days for active participation were Monday and Saturday with third closest day (Tuesday) running 23% lower.

Users logged in to the system more frequently on Mondays, Wednesday and Sundays. This pattern can likely be explained by the fact that our class held lectures on Mondays and Wednesday afternoons. Homework deadlines for our course commonly occurred on Fridays, when involvement with REGIS was at its lowest.

Guesses peaked on Mondays and Saturdays. Participation rates drop continuously as the week progresses until the weekend sets in; since student workloads often increase as the week goes on and then decrease over weekends, there appears to be a correlation between how often students make guesses on Codepath and how much other work the students have at that time. This hypothesis is somewhat intuitive -- students would logically put off using Codepath during demanding periods given that our course doesn't place any participatory requirements on student involvement with the tool.

There seem to be two different behavioral cases that appear in the usage graph in Figure 12. The first of these, present on Tuesday and Saturday, are the cases with a high number of guesses per login. This suggests that highly active users were participating on these days, and that there were likely fewer users who logged in and left without answering any questions. The second behavioral case is the opposite, most notably present on Wednesdays and Fridays, where students cumulatively logged in more than they guessed, meaning that there was an average of less than one guess per person.

In-class performance

We were also interested in seeing how students' grades were affected after becoming Codepath users. We administered two exams in the course that impacted the student's grades and contained no direct material from any of the Codepath exercises. The first exam was administered before Codepath was announced to the class and the second exam was given after Codepath was available for approximately 50 days. The exams contained questions on both course readings and programming, but the results reported below are based only on the questions about programming-related topics.

The users who created accounts with Codepath were evenly distributed between the top and bottom 50% of scores from the second test. Only three of the top ten students and two of the bottom ten students had accounts on Codepath. This suggests that Codepath did not appeal more to either underperforming or high-performing students, but the distribution instead suggests that students near the middle of the performance curve had the highest enrollment and guess rates.

The students who created an account on Codepath improved an average of 12.667% between the first and second tests, compared to 8.906% for the class as a whole. Many of these users, however, did not actively participate. The students who provided at least one guess to a Codepath exercise showed an improvement of 9.936%, which is only marginally better than the class average. This suggests that a student's willingness to join Codepath may be generally indicative of the effort they're willing to invest, but those students don't necessarily use Codepath as a means of studying or improving their understanding of programming topics.

User Category	**Improvement between tests**
Full class	8.906%
All Codepath users	12.667%
Codepath users w/ at least one guess	9.936%
Codepath users w/ at least one *correct* guess	9.412%

Table 1: Those who used Codepath and those who did not showed similar levels of improvement between exams.

This result is not particularly surprising due to the low participation rate among many users. The effect, if any, would likely be far more pronounced with more higher levels of participation and may be more measurable in future semesters. It is unclear, however, whether any improvement would be correlated with the use of Codepath or simply an indicator of a more involved student, but we plan to continue monitoring this metric in future work if participation rates improve.

Student Feedback

We administered an optional survey at the end of the semester open to all students who logged in to Codepath at least once during the trial period and received eight responses. Students reported spending 5 – 30 minutes per problem that was answered, and everyone who responded claimed to have answered at least one problem.

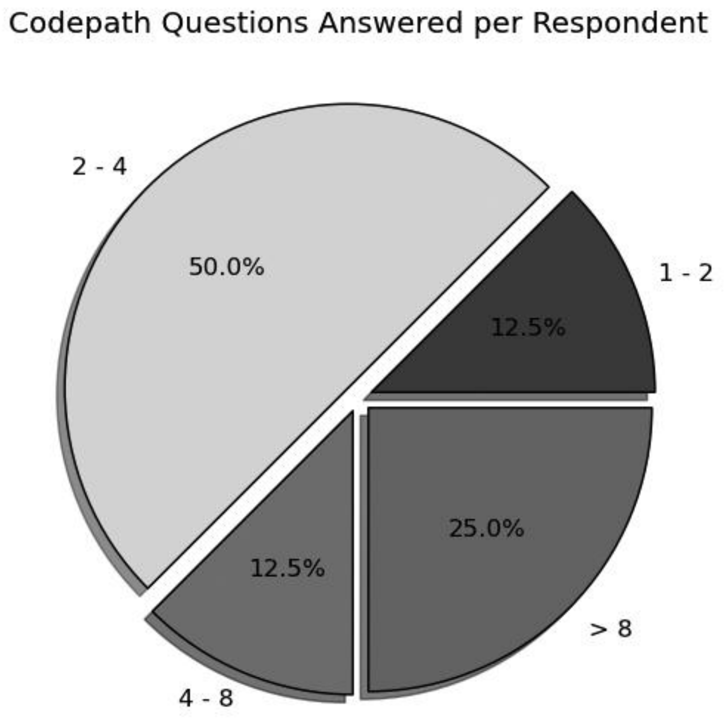


Figure 13: All of the eight respondents answered at least one Codepath question during the trial period. This suggests that the sample was unrepresentative of the class as a whole but provides potentially useful insights nonetheless.

We asked how students thought a tool like Codepath could best be integrated into the course, and the most common recommendation (stated by two of the eight respondents) was to somehow incorporate it into the grading structure of the course. Other responses were scattered but often focused on question-specific issues or comments on the user interface, which has since been replaced by the flash card UI.

Out of the eight respondents, four of them explicitly indicated that they think it would be beneficial to have Codepath exercises be a required part of the course; one explicitly disagreed with this statement. Five of the eight respondents indicated that they would have done more Codepath problems if they'd had more time during the semester, and three responded with a "maybe."

The evidence from this survey, though sparse, confirms our belief that integrating Codepath as an open but required portion of the course is worth attempting, and may lead to an improved student experience and higher degree of student comprehension.

FUTURE WORK

REGIS is a proof-of-concept tool that builds a foundation for sharing problems from many contributors. Many of the mechanisms are likely imperfect and could be improved or replaced in the future. Some of the key areas for improvement are listed below with some proposed directions for how each one might be improved.

Implementations in alternative social contexts

The engagement results for our implementation of REGIS were relatively low, but we suspect that this is a result of the social and pedagogical context that it was presented in (optional with limited integration in classroom). The primary limitation has likely been that students did not have enough time in their schedules for optional activities. In future semesters we plan to experiment with a number of different implementation styles now that the tool is relatively mature and the existing question database has a workable number of problems in it. Our next experiment will likely involve requiring students to answer a small number of self-selected questions each week; submitting questions will likely be left as an optional activity that may result in some small form of participation-based extra credit.

Question sharing mechanisms

Questions currently need to be shared manually by transferring solver files and question text between deployments; ideally it would be simple to discover new questions and import / export questions from a deployment. This would significantly decrease the workload for question generators and would facilitate the simple sharing of high-quality questions among organizations and institutions.

We have designed our question provider module with CourseSharing in order to enable the sharing of questions, but there's still an open question of whether the sharing of questions could take place in a straightforward way from within the REGIS interface.

Some important issues arise in terms of securing deployments when arbitrary code like solvers and term definitions are copied over from remote (and potentially untrusted) sources. The design and facilitation of this process could easily be a project within itself. However, it would likely lead to a significant reduction in effort and a corresponding boost in quality of review materials among leagues.

Inter-league sequencing

Effective sequencing requires a significant amount of data in order to properly predict both the ability of users and the difficulty of questions and categories of questions. If questions are shared among larger populations, either copied between leagues or shared using the general mechanisms described directly above, the sample size for an individual question or category could be increased significantly.

In addition, if users join multiple leagues with high enough frequency then it may be possible to predict the difficulty of certain questions based on the ability of the same user in other leagues; it seems reasonable to expect that a user with high performance in one league would have a greater-than-average chance of having high performance in another league, especially if the leagues are topically similar. It is currently unclear whether this hypothesis is true due to a shortage of data but it would be an interesting avenue of exploration if REGIS or a similar project could generate useful data.

Interest prediction

Since one of the primary features of REGIS is the ability for users to submit questions that can be viewed by others, performing some sort of intelligent content recommendation becomes important in order to help users navigate the options that are available to them. The information gathered from the individual question feedback could be used as one of the inputs to this predictor.

The body of academic literature around interest-based content recommendation is vast, but very little work has been done specifically in the context of educational content. It seems reasonable to expect that well-known predictive techniques like collaborative filtering and regression would be effective in predicting interest levels for educational content as well; if an effective method could be applied then this could become an important component of REGIS's content sequencing algorithm, or a separate component of the system that could provide customized problem recommendations in the user's profile. Incorporating interest measures as well as the current difficulty measures would likely provide a more intriguing sequence and could help students explore topics in the contexts of other disciplines.

Team support

Planned future work involves creating a module for Regis that would allow participants to form teams of any size (including one) and challenge other teams to competitions directly. This module could be activated or deactivated on a per-league basis. Students could receive credit for their success based on the relative skill level of their team and the opposing team using an appropriate ranking mechanism like the Elo rating system, a method often applied to competitive chess ladders [46].

League-wide operating modes

Possible future work involves creating generic operating modes that can be easily customized by the league administrator. The various system mechanics (question release frequency, exam mode status, instructor / peer manual review mode flag, etc) could be modified on a site-wide or league-wide basis in order to increase the applicability of REGIS to other use cases.

CONCLUSION

Since its launch, Codepath has attracted a significant portion of the students in our course and has confirmed that students are interested in using the tool. Despite the fact that participation rates were low, we believe that REGIS is still a valuable classroom tool that can be improved further with additional investigation and development. Some of those improvements, outlined in the previous section, could significantly impact its adoption rates among both teachers and students. Alternative classroom integration strategies could also help determine whether the tool was useful to students.

Further work is needed in order to make the tool widely distributable but REGIS has come a long way towards addressing an identified classroom need. In addition, feedback from students (users) and other teachers has helped identify important future directions for the tool.

We plan to continue our work with REGIS to iterate on the tool with other users, potentially in a high school setting. We would also like to launch the tool so that leagues could be created easily, opening up availability to users in informal learning settings. In addition, we will be using REGIS in both online and in-person iterations of our current course in order to further develop the tool and improve student learning and our course's overall appeal.

REFERENCES

[1] D. Merrill, "First Principles of Instruction," *Educational Technology Research and Development*, vol. 50, no. 3, pp. 43-59, 2002.

[2] D. Garcia and B. Harvey, "CS10: The Beauty and Joy of Computing," 2012. [Online]. Available: http://inst.eecs.berkeley.edu/~cs10/.

[3] M. Clancy, N. Titterton, C. Ryan, J. Slotta, and M. Linn, "New Roles for Students, Instructors, and Computers in a Lab-based Introductory Programming Course," in *SIGCSE*, 2003.

[4] A. Krapp, S. Hidi, and K. A. Renninger, "Interest, Learning, and Development," in *The Role of Interest in Learning and Development*, K. A. Renninger, S. Hidi, and A. Krapp, Eds. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1992, pp. 3-25.

[5] E. Deci and R. Ryan, *Intrinsic Motivation and self-determination in human behavior*. New York: Plenum Press, 1985.

[6] E. Deci and R. Ryan, "The Support of Autonomy and the Control of Behavior," *Journal of Personality and Social Psychology*, vol. 53, pp. 1024-1037, 1987.

[7] J. Connell, "A New Multidimensional Measure of Children's Perceptions of Control," *Child Development*, vol. 56, pp. 1018-1041, 1985.

[8] S. Harter, "The Perceived Competence Scale for Children," *Child Development*, vol. 53, pp. 87-97, 1982.

[9] R. Ryan and J. Connell, "Perceived Locus of Causality and Internalization: Examining Reasons for Acting in Two Domains," *Journal of Personality and Social Psychology*, vol. 57, pp. 749-761, 1989.

[10] M. Prenzel, "The Selective Persistence of Interest," in *The Role of Interest in Learning and Development*, K. Renninger, S. Hidi, and A. Krapp, Eds. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1992, pp. 71-98.

[11] R. Schmidmaier, R. Ebersbach, M. Schiller, I. Hege, M. Holzer, and M. Fischer, "Using electronic flashcards to promote learning in medical students: retesting versus restudying," *Med Educ*, vol. 45, no. 11, pp. 1101-1110, 2011.

[12] A. Tan and T. Nicholson, “Flashcards revised: Training poor readers to read words faster improves their comprehension of text,” *Journal of Educational Psychology*, vol. 89, pp. 276-288.

[13] H. Stutz, “Flashcards: Fast and Fun,” *American Association of Teachers of Spanish and Portuguese*, vol. 75, no. 5, pp. 1323-1325, 1992.

[14] M. Houston and L. Lin, “Humanizing the Classroom by Flipping the Homework versus Lecture Equation,” *Proceedings of Society for Information Technology & Teacher Education International Conference*, pp. 1177-1182, 2012.

[15] D. Rowntree, *Exploring Open and Distance Learning*, 3rd ed. London: RoutledgeFalmer, 1992, pp. 29-31.

[16] P. Khan and K. O’Rourke, “Understanding Enquiry-Based Learning,” in *Handbook of Enquiry and Problem-based Learning Irish Case Studies and International Perspectives*, T. Barett, I. M. Labhrainn, and H. Fallon, Eds. 2005, pp. 1-12.

[17] A. Kamenetz, *DIY U: Edupunks, Edupreneurs, and the Coming Transformation of Higher Education.* White River Junction, VT: Chelsea Green Publishing Company, 2010.

[18] H. Finn, “Watching the Ivory Tower Topple,” *Wall Street Journal*, 23-Mar-2012.

[19] “Project Euler.” [Online]. Available: http://projecteuler.net.

[20] N. Parlante, “Coding Bat.” [Online]. Available: http://codingbat.com.

[21] Google, “Code Jam.” [Online]. Available: http://code.google.com/codejam.

[22] “Memorize.com.” [Online]. Available: http://www.memorize.com.

[23] “Course Sharing.” Berkeley, CA, 2012.

[24] MIT and Harvard, “edX,” 2012. [Online]. Available: http://www.edxonline.org/.

[25] Khan Academy, “Khan Academy.” [Online]. Available: http://www.khanacademy.org/.

[26] Programr, “Programr.com.” [Online]. Available: http://www.programr.com/.

[27] MasteringPhysics, “MasteringPhysics.” [Online]. Available: http://www.masteringphysics.com/.

[28] “Turing’s Craft.” [Online]. Available: http://www.turingscraft.com/.

[29] P. Brusilovsky, “A Framework for Intelligent Knowledge Sequencing and Task Sequencing,” *Intelligent Tutoring Systems*, vol. 608, pp. 499-506, 1992.

[30] F. Lord, *Applications of Item Response Theory to Practical Testing Problems*. Mahwah, NJ: Erlbaum, 1980.

[31] Blackboard, “Blackboard.” [Online]. Available: http://www.blackboard.com/.

[32] Sakai Foundation, “Sakai CLE.” [Online]. Available: http://www.sakaiproject.org/.

[33] “Moodle.” [Online]. Available: http://www.moodle.org/.

[34] “Piazza.” [Online]. Available: https://www.piazza.com/.

[35] “Build Your Own Blocks (BYOB).” [Online]. Available: http://byob.berkeley.edu/.

[36] K. Heimerl, B. Gawalt, and K. Chen, “Communitysourcing: Engaging Local Crowds to Perform Expert Work Via Physical Kiosks,” *Proceedings of SIGCHI*, 2012.

[37] “jQuery: Write Less, Do More.” [Online]. Available: http://www.jquery.com/.

[38] “Backbone.js.” [Online]. Available: http://documentcloud.github.com/backbone/.

[39] C.-M. Chen, H.-M. Lee, and Y.-H. Chen, “Personalized E-Learning System Using Item Response Theory,” *Computers & Education*, vol. 44, pp. 237-255, 2005.

[40] J. Breese, D. Heckerman, and C. Kadie, “Empirical Analysis of Predictive Algorithms for Collaborative Filtering.” Microsoft Corporation, Redmond, WA, 1998.

[41] L. Adamic and B. Huberman, “Zipf’s Law and the Internet,” *Glottometrics*, vol. 3, pp. 143-150, 2002.

[42] J. Nielsen, “Participation Inequality: Encouraging More Users to Contribute.” 2006.

[43] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design Lessons from the Fastest Q&A Site in the West," *SIGCHI*, 2011.

[44] A. Kittur, E. Chi, B. Pendleton, B. Suh, and T. Mytkowicz, "Power of the Few vs. Wisdom of the Crowd: Wikipedia and the Rise of the Bourgeoisie," *World Wide Web*, 2007.

[45] F. Fu, L. Liu, and L. Wang, "Empirical Analysis of Online Social Networks in the Age of Web 2.0," *Statistical Mechanics and its Applications*, 2008.

[46] A. Elo, *The Rating of Chess Players, Past and Present.* Ishi Press, 1978.

[47] "Code Academy." [Online]. Available: http://www.codecademy.com/.

APPENDIX A: STUDENT SURVEY RESULTS

The following table contains the numeric results of each poll, as well as the precise wording for each of the elements on the poll.

Proposed content addition	**Fall 2011 results (n=173)**	**Spring 2012 results (n=168)**
"Test yourself" mini-quizzes	5 (3%)	17 (10%)
Mini-programming challenges	64 (37%)	73 (43%)
Tree-structure interface for lectures	7 (4%)	17 (10%)
1080p high definition archived lectures	5 (3%)	10 (6%)
"Instructor takes the class" videos of us doing labs, HW, exams	92 (53%)	51 (30%)

APPENDIX B: TERMS LIST

Adding terms to REGIS is relatively straightforward but requires some programming. The following terms have been implemented in order to generate the required fields for our question corpus. Each term is written in bold and followed by any parameters that the term accepts. Optional parameters will appear in square brackets.

Several terms perform selections on items or sets of items stored in server-side files. Individual items in a file are partitioned by newline characters. Sets are partitioned by placing an empty line between the previous set and the successive set.

`caesar` `phrase shift_amount`

> Performs a Caesar shift on `phrase` by shifting each character by the `shift_amount` spaces after it in the alphabet.
>
> Example: casesar house 3 => krxvh

`chooseset` `filename`

> Reads a file accessible by the REGIS server and extracts a set randomly from the list of all available sets.

`choosestr` `filename`

> Reads a file accessible by the REGIS server and extracts a single item from the list.

`even` `min max`

> Produces an even number between MIN and MAX.

`formatdate` `month day year`

> Generates the text version of the numerical date provided as input.
>
> Example: formatdate 1 22 1987 => “January 22, 1987”

`link` `data`

> Stores DATA in a user-specific file and generates a URL for accessing the file. This URL can appear in the question body and will open a text file in the user’s browser when activated.

`mixset` `filename`

Reads a file accessible by the REGIS server and extracts a set randomly from the list of all available sets. It then randomly scrambles that set before returning it.

This term is like chooseset but randomly shuffles all of the items in the chosen set.

`numstring` `length`
: Generates a string of random numbers LENGTH digits long.

`num` `min max`
: Generates a random number between MIN and MAX.

APPENDIX C: SOFTWARE AVAILABILITY

REGIS is freely available online under a GPL 2.0 license. The latest source code is available from https://github.com/luke-segars/regis. Project updates and patches will also be available from this URL.