A Service Learning Practicum Capstone

Aaron Bloomfield
University of Virginia
aaron@virginia.edu

Mark Sherriff
University of Virginia
sherriff@virginia.edu

Kara Williams
Center for Nonprofit Excellence
kwilliams@thecne.org

ABSTRACT

We present the design and execution of a Service Learning Practicum (SLP) course sequence intended to be year-long capstone for computer science seniors. Students are teamed into groups of six, and develop software for local nonprofit organizations. In addition to the structure of the course, we describe the challenges faced (legal, organizational, etc.), student perceptions via survey results, and provide a number of suggestions for other institutions who are looking to create a similar course sequence. At the end of the capstone experience, the customers are provided with working software that meet their current needs.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Management, Human Factors, Legal Factors

Keywords

Capstone courses, service learning, project based learning

1. INTRODUCTION

Keeping students involved and interested in long and complex projects can be challenging for even the most talented instructor. Toy projects created for pedagogical value can be useful, but students can often see beyond the assignment and know that what they are actually working on will not have any intrinsic value in the future. Layman and Williams might have put it best when they argued to just "make assignments meaningful" [3]. One way to make an assignment meaningful is to provide a focus for the assignment where the students are helping others.

We present this experience report describing a Service Learning Practicum (SLP) that was developed and implemented at the University of Virginia.

SIGCSE 2014 Atlanta, Georgia USA
ACM 978-1-4503-2605-6/14/03 ...$15.00.
http://dx.doi.org/10.1145/2538862.2538974.

The first of two goals of the SLP is to teach students about development of large software projects. These are concepts that are taught in many courses, but due to the nature of collegiate classes, few courses allow for the development of large projects in groups. In the SLP, students are teamed into groups of six students with one adviser or mentor. The students develop a year-long project in the same manner as a professional development company. The skills, knowledge, and concepts that they learned in their various classes can be put to use, while learning aspects of teamwork, customer interaction, and management skills.

Developing a project that is "not meaningful" can make the students feel that their time is not being respected as burgeoning professionals. This begets the second goal of the SLP: to write quality software for nonprofits in our community. Working with nonprofit organizations allows the students to interact with a real customer, developing real software with a real purpose. Furthermore, the software will have a measurable, positive impact on our community.

The result of the SLP is a capstone course that combines a significant software engineering project – including customer interaction – with a service learning experience.

We are certainly not the first to present a software engineering based capstone, nor the first to present a service learning course with a software engineering focus. However, we feel that we have synthesized the results of previous literature into the successful creation and implementation of our SLP, and we provide a number of structural suggestions and lessons learned that will be of use to a faculty member who is attempting to implement a similar practicum at his/her own institution.

2. RELATED WORK

One of the main aspects of service learning courses is in the idea that the project has intrinsic social value and meaning that the student can believe in. Layman and Williams examined assignments from CS1 and software engineering courses and found that computing instructors still need to look to meaningful assignments for their students [3]. They note that often the only change necessary for making an assignment social relevant is a wording or domain change for a particular computing problem. In our case, students worked mainly on database-driven web applications. These could have been for nearly any domain that the instructor could think of. However, by bringing in nonprofit organizations, the same assignment (creation of a database-driven web application) suddenly becomes something that the students can more easily get behind and be excited for. We

contend that, while coordination is required to get nonprofits involved in the projects, the results are worthwhile.

There exist many different models for service learning courses in computer science. Many are single semester course offerings where a course project – or perhaps a semester-long project – is the service learning project itself. Some of these models were discussed at a recent SIGCSE panel [5].

Carnegie Mellon University has developed a model where students are consultants, and work with a nonprofit to help them solve their technical problems. Their approach is more systems engineering based rather than software based, and has achieved remarkable success – in the 14 years of the course's existence, it has had close to 400 partnerships [4].

Egan and Johnson report on incorporating service learning into an introductory CS course, as opposed to a capstone-type course as we implemented [2]. One major advantage that they saw from their course structure was an increase in retention rate of students from CS1 to CS2. This follows from the ideas of Layman and Williams that when assignments are meaningful, students are more engaged and are more likely to continue. However, other students had a less positive experience, as they reported additional homework and pressure on completing the project on time.

Traynor discusses two models for service learning [6]. The first is an early computing course where the students work to reduce the "digital divide" separating generations. Each student is grouped with a senior citizen and a 3rd grade student, and helps each of them with learning to use computers and the Internet. The second model is an upper-level HCI course where the project is to aid local nonprofits. Service examples of the latter include creating a website, and rewriting a computer manual into a format easier to understand by a non-technical person.

Many other institutions have used single semester courses where a project – either one of many projects or a semester-long project – is a service learning project. Since we are presenting the results of a two-semester service learning capstone sequence, and not a single semester course, we do not attempt to properly catalog such single semester offerings.

3. COURSE DESIGN

Our Service Learning Practicum was run last year (2012–2013) as a two semester, fall–spring academic year sequence (39 students, 7 projects). This year (2013–2014) is an ongoing iterative evolvement of last year (43 students, 7 projects). This article presents the results of the completed course (2012–2013), as well as describing some of how we modified it for the current (2013–2014) academic year.

The course is designed as a practicum-based course, and not a lecture-based course. Homework assignments for the course are made up of the multiple deliverables for the individual student projects. There were no exams, although there was an final project required of each student.

The course meets three times a week. On Mondays, we typically have a combined lecture for all the teams on a topic that we considered important for this course: requirements analysis, customer interaction, development methodologies, etc. Some Monday lectures are half-lecture, half group break-out session. Wednesdays are customer meeting days, where teams meet in person with their customer. These meetings often take place during the scheduled class period at other places around campus, but students can meet at another time during the day as well. Fridays are reserved for team work days. Student teams use the Friday class period for meetings with their teammates on particular aspects of the project or to get more in-depth help from the course instructor.

For the development of the projects themselves, we use features from a number of agile development methodologies. From the Rational Unified Process, we use the concept of inception and some of the documentation. From Extreme Programming we use the concept of the customer team member, collective ownership, short cycles, and continuous integration. From Scrum we use the ideas of regular scrum meetings, backlogs (both product and sprint), and burndown charts. We use Redmine, an online project management system, to help with project organization.

Our sprints or iterations are two weeks in length, and end on alternating Mondays. The customer meetings typically occurred on the second day of a sprint, which allowed the customer to view the results of the previous iteration and arrive with comments, questions, suggestions, and – all too often – requirement changes.

4. COMMUNITY PARTNERS

A practicum of this size is not feasible alone. In addition to the nonprofits that we developed the systems for, there were others in the community that we worked with.

4.1 Partner Nonprofit

While the course instructors have had some experience with local nonprofits (serving on boards, previous volunteer service, etc.), being as we are computer science faculty, nonprofit work is obviously not our primary job. Our contacts, knowledge, and experience with the local nonprofit community were not sufficient to allow us to generate the necessary response for the SLP. Thus, we needed to reach out to the nonprofit community.

We were fortunate to be able to partner with the Center for Nonprofit Excellence (CNE), a local nonprofit whose purpose is to strengthen other local nonprofits. CNE is "the area's primary resource for nonprofit management, leadership and collaboration" [1]. CNE has contacts with most of the local nonprofits, and thus has a feel for the type of projects that can best be of help – and who could most benefit. Indeed, when soliciting projects for the current year, a single email generated almost 50 nonprofit project requests. CNE performs a technology survey of nonprofits, and the results of this survey helped us to identify which nonprofits would benefit the most from a custom-developed system.

4.2 Mentors

Students working on these projects need advice and guidance, as they never have been involved on a software development project of this size. As the size of the course grows, it is not feasible for the course instructor to be at all the group meetings – for one reason, many of them are likely to happen at the same time but in different locations. This gave rise to the concept of a mentor: a professional software engineer with years of experience in software development. One mentor is assigned to each group of six students. A mentor attends the group meetings with the customer and generally keeps up with the development of the system. A mentor's job is to help advise the students, not to develop the software. Similarly, a mentor is not meant to be a replacement or a surrogate for the course instructor – s/he is

somebody in an advisory role who has a lot of knowledge and experience in software development, and is interested in helping out with this type of project. Having a mentor act in an authoritative role could potentially introduce other problems – including a situation where a non-university individual has control over student work – as well as possibly negatively affect the student-mentor relationship.

The mentorship idea has a number of advantages. It allows the students to benefit from the years of experience from the mentor. It lets students get to know somebody in the software business, and helps the mentor get to know students who may soon be looking for employment. Lastly, it allows professional software developers to help out a local nonprofit while using their expertise.

We were concerned that we would not be able to find enough mentors, even though there are many local software development companies. However, one email announcement yielded enough mentors in only a few days, and then we had to start turning others away. At a meeting of the mentors and the course instructor, they were all very enthusiastic about being involved in such a project, and at being able to help their community while using their specialized training.

We feel that adding computing professionals in both the service and the educational aspects of the course makes our version of a service learning experience that much more compelling. Students get to work with a computing professional who has years of experience, and who can help the students learn to work effectively with customers. Computing professionals similarly get to have an outlet to use their skills in a socially relevant way. These partnerships can then lead to more community connections and even to more potential projects and mentors.

5. PROJECTS

The projects for our service learning course came from organizations of varied purpose, location, and outreach. In this section we discuss the challenges of finding the right mix of projects for students.

5.1 Requirements

Project selection was a challenge for this practicum. A project must be complex enough to serve as a year-long capstone for six computing seniors, yet be able to be completed within the context of a single academic year. Duplicating an existing piece of software did not seem like a viable use of our time. For example, one nonprofit wanted a system to manage collaborative document editing, and we referred them to Dropbox (which is fulfilling their needs quite well).

The course instructor met with various nonprofits during the summer prior to the academic year to help clarify the details of each project. This enabled a reasonable estimate for a properly complex and scoped project to emerge. The students are presented with a general requirements document for each project describing what is desired. Such a document is typically 3-5 pages, and is meant to get them started on understanding the technical details of the project.

Properly scoping a project can be a large concern for this type of course. Indeed, one of the biggest problems we encountered in the first instance of the course was trying for too large of a scope of features to include. To help mitigate this, each requirements document had three categories of requirements: minimum, desired, and optional. The minimum requirements are just that – the minimum requirements necessary to have a system that is of value to the nonprofit. The desired requirements are what we would like to have accomplished by the end of the academic year. The optional requirements include features to include if we choose too small of a scope. The selection was done so that our best estimate is that the desired requirements are feasible within a single year. We were highly accurate in this regard this past year. The students are told that they need to have a prototype of the system with all the minimum requirements at the end of the fall semester, and the full system (with all the desired features) deployed at the end of the academic year. We realize that the original set of minimum and desired requirements may not be feasible within the time allowed, and thus are flexible if those requirements categories need to change.

5.2 Project Restrictions

We specifically avoided certain organizations when we solicited projects. Religious organizations were acceptable, as long as the purpose of the charity – and thus the system to be developed – was for a secular charitable cause. Separation of church and state – as we are a state institution – dictates the necessity for this, as well as the fact that it may make some students uncomfortable otherwise. For example, one nonprofit organization we worked with, Appalachia Service Project, is such a religious organization with a secular charitable purpose.

We also decided that the nonprofit must have a charitable goal as its primary purpose. Many do not, such as athletic leagues, political organizations, fraternities and sororities, etc. Our definition of a political nonprofit here was one that contributed to one or more political candidates, either directly (through campaign donations) or indirectly (through activities such as lobbying).

5.3 Project Examples

In the two years that the SLP has existed, there have been 14 projects, 7 each year. We only describe two projects here; the full list can be found online[1].

Scheduling is a common area in which we found many projects. There are many organizations who have such specific scheduling constraints that there are no free or viable commercial software systems that are able to perform this scheduling. Since these are nonprofits with limited budgets, a system may not viable simply be because it is expensive.

The Virginia Institute for Autism works with autistic students. Teachers are assigned, generally one-on-one, to individual students for six periods throughout the day; teachers generally change students each period. There are numerous constraints on this scheduling: student and teacher absences change who is working with who, teachers can only work with students for which they are trained, lunch requires other teachers to cover their classes, etc. A daily schedule takes six people 30 minutes (i.e., 3 person-hours) to create. With the system in place, it took only one person about 30 minutes to create the schedule, as many pieces of data – such as who is absent that day – still needed to be entered.

Another common topic for projects was website-based or smartphone-based systems. There was sufficient complexity when such things as user accounts, data entry, and custom searches were features that were included in the system.

Habitat for Humanity of Greater Charlottesville builds residences for low-income individuals. Once a house is con-

[1] `http://www.cs.virginia.edu/~asb/slp/`

structed, Habitat generally hands off the management of the house to the new owner(s). However, there is interest in keeping track of the houses in a community that have been built. Habitat was keeping track of this data in many disparate places. The system that was developed allowed for a single place to enter the data, and was able to pull results from the data that was not possible in their previous setup. This has saved them significant time, especially with their reporting requirements, as the system can pull the necessary data from the common database very quickly.

5.4 Students and Projects

Students are allowed to specify their project choice preferences. Since the population size for this is relatively small ($n = 39$ students for the 2012-2013 academic year), the matching and project assignment is performed manually.

Once the student group is assigned, they have to choose a student to be the project lead. The lead performs much of the coordination for the team, such as scheduling, keeping the meetings moving along with their agenda, keeping the course instructor updated with the overall team progress, etc. It is an administrative position, not an authoritative one. We feel that this is a valuable experience for students, and that all of them should experience it at some point – so for that reason, the project lead rotates through each of the students throughout the year; this allows each student to be the lead for one or two months.

As much as possible, we prefer that team decisions are made by consensus. Ultimately the customer gets to choose what features should be implemented, what it should look like, etc. But the team can choose how it is to be developed in software. Recall that we did not want the mentors to act in an authoritative role, as described above.

We require the students to track their time spent on the project. This allows us to determine, at the end of the year, how many hours were devoted to nonprofit software development – a figure that is particularly useful when analyzing the impact of the SLP. To prevent hour inflation, we guarantee that a student's grade will *never* be evaluated based on the hours they submit. The reality is that if a student is not contributing sufficiently, there will be ample other ways to discern this fact.

6. LEGAL ISSUES

In consultation with our university's general counsel and the provost's office, we worked on developing an appropriate wording for a license with the nonprofit organizations. We wanted a nonprofit to be able to use the software forever, but university policy, consistent with US copyright law, dictates that students own the code that they develop. Most of the projects are developed using interpreted or scripting languages (PHP and Javascript being the most common), so the source code *was* the software. Regardless, one of the deliverables to the nonprofit would be the source code, even on a compiled platform; the other deliverables being the working system and appropriate documentation.

The result was a delicate balance between preserving the students' rights to the code they developed while also allowing the nonprofits to use it. The nonprofits would agree to a perpetual non-commercial, non-distributable, non-revocable license to the software. In addition, they can modify the source code. A summary of the license's terms follows:

- Perpetual: the nonprofit can use this software forever.
- Non-commercial: the nonprofit may not sell the software, nor make money off of charging individuals for using the software (i.e., it must be "free" to use). Some of the software may not be for public distribution – such as an internal scheduling system – but any public-facing part must be able to be used free of charge.
- Non-distributable: the nonprofit may not distribute the software. University policy (and US copyright law) dictates that undergraduate students own any code that they develop for a course (this is different than if a student is paid for the software development, such as a graduate student or a paid undergraduate research assistant). Because of this policy, the nonprofits do not own the code, and thus they cannot distribute it.
- Non-revocable: the license cannot be taken back from the nonprofit by the students or by the university.
- Modifiable software: the nonprofit would gain the actual source code; as mentioned above, it is often the case that the software *is* the source code, but even with a compiled system, the nonprofit would receive the source code. They are allowed to modify it, such as by hiring a developer to add features or remove bugs in the future; this modified version also cannot be distributed or sold.

Note that the agreement does not lessen students' rights at all; it only grants some (limited) rights to the nonprofit. While we could not require students to sign it, we did not encounter any significant opposition with the students signing the agreement. As the students owned the software, it was their decision as to what to do with it – release it under an open source license, attempt to sell it, etc.

As the course instructors, we developed some software for the projects, typically in the form of tutorials and source code examples. A conflict of interest situation arose here. We were developing software as part of our salaried job, giving it to students, and potentially claiming ownership of this code – which could imply partial ownership of the system, and thus could allow us to make a claim to some monetary gains if the students sold their code. For this reason, the course instructors relinquished any rights to the code that we provided to the students. Mentors do not develop any source code, and thus do not need to agree to this license.

The other legal concern that we encountered was the use of existing source code. There are many licenses that exist that both open source and free software are distributed under. Some licenses – specifically the GNU Public License (GPL) – require that any software developed that uses that GPL code be publicly distributed under a GPL-compatible license (which is usually just the GPL itself). This meant that students could not use any GPL software unless they were willing to publicly release their *entire system* under the GPL. Note that this does not apply to linking to a library compiled under the GPL – those libraries are generally covered under the LGPL license, which allows for linking without release of source code. While many people tend to ignore these licensing issues, because we were developing real software for other organizations, it was necessary to carefully adhere to the software license requirements.

There are many licenses that do not have the so-called "viral" licensing restrictions of the GPL. Such licenses – such as the Apache License, the MIT License, and the Mozilla Public License – allow for use of such code in a system without

releasing the resultant source code to the public; however, one must typically include credit to the source code used (how this is to be done varies by the license).

7. LESSONS LEARNED

We felt the course was a success in most regards – students learned a lot, we delivered quality software, the nonprofits were happy with the results, and the community-student interaction was allowed to flourish. However, there are some important lessons learned that we share here.

Customer involvement. The customers were, generally, not familiar with software development. One of the aspects of Extreme Programming is the idea of the customer team member. We attempted this last year, but we did not sufficiently set the expectations of what is needed from the customer ahead of time. Some customers were very involved, and their projects flourished. Other customers put in the minimal amount of work necessary, and their projects did not fare so well. In the current year, we have created a customer agreement, which outlines what is expected of the customer. In particular, we mention three aspects: biweekly meetings, providing regular feedback, and early testing of the system. We feel that this agreement, and the associated sufficient buy-in from the customers, will solve the vast majority of the issues that arose last year.

Project start-up and transition. There is a large ramp-up at the beginning of the year, as the students acclimate to the project and the technologies used. In particular, learning the software framework will take them some time. In the second year of this course, we had them start on the projects two weeks into the semester. This year we are having them spend two more weeks (for a total of four) working on individual assignments meant to teach them the framework. This lessened the number of iterations by one, but allowed them to have a better understanding of how to design the system for when they start on their group project.

Towards the end of the project, there is the deployment, maintenance, testing, and documentation – all tasks that take non-zero time. We started planning for deployment as soon as the spring semester started, and it was still not enough time for two of the projects. This year, we worked with the customer to determine reasonable expectations before the deployment phase to hopefully alleviate this issue come this spring.

Maintenance Connected with the project transition lesson is the proper way to handle project maintenance post-release. At the end of the course, the nonprofits that we developed the projects for were quite happy with the results. We were generally able to complete the requirements specified. However, there is little means for maintenance, as the students will graduate and move on with their lives shortly after project delivery. We did ask the students at the end of the semester if they were willing to be contacted about helping to maintain the project in the future, and 79% of the class indicated they would (and at least 4 from each group). Thus, it is critical that the nonprofits have some means to maintain the system. This could be through a volunteer in the community who is willing to take over this task; however, finding one skilled in the framework is a challenge. Local companies may be willing to take this over as well. On some projects, the students themselves were willing to be contacted again to help maintain the system, as they felt emotionally invested.

Requirement solicitation We quickly realized that the requirements needed to have some flexibility. Thus, we split the requirements into "minimum", "desired", and "optional", as described above. This allows us to focus on the core system first, the ideal system next, and then expand if they end up with extra time later.

8. RESULTS

At the end of the spring 2013 semester, we had the students fill out an extensive survey detailing their perception of the class and the various aspects therein. The survey consisted of both text fill-in answers as well as 5-point Likert questions. All the Likert questions were on a 1 to 5 scale. For the course-wide questions, $n = 38$. The results are shown in Table 1. For the results for a particular group (not shown in the table, but described below), $n = 6$.

All the projects had some requirement changes, and the number of changes was rated at 3.50. In one project, the customer changed her mind on the first day, and in another project the goal was to develop many of the requirements during the year. If you take these two projects out of consideration, then the overall rating of how many requirements changed was 3.10, which is very close to the "neutral" answer, meaning that it was a reasonable amount of changes for a project like this. The requirements changes were considered somewhat predictable by the students (3.13).

Satisfaction with the customers by the students was high (4.11), although one group was much less satisfied with their customer (3.00). We knew that this particular customer was a problem during the school year. If one does not consider that one group, then overall customer satisfaction increases to 4.35. Interestingly, this one group with low customer satisfaction was also the group that had the overall least project satisfaction (4.17), even though they were still mostly satisfied with their project. That project ran into many technical difficulties, so there were certainly other factors contributing to their lower project satisfaction.

Satisfaction of the mentors by the students was a bit lower, at 3.89. We attribute this to a number of factors. One was a lack of a clear set of mentor expectations. Another is that the mentors often were busy, and thus unable to attend the group meetings. However, the feedback that they provided was considered very helpful (4.18), and the students generally recommended the customers again (4.16), although there was much variation in the mentor survey results.

Overall project satisfaction was high (4.48), and students felt they would likely have chosen the project again, if they knew then what they know now (4.03). We were pleased to see that the groups worked together quite well (4.47).

We did not perform a formal survey of the customers. However, we do have some anecdotal evidence: each customer was contacted about their overall satisfaction, and all the responses were positive.

Question	Likert 1-5 scale (1 is a "bad" result, 3 is neutral, and 5 is a "good" result)	Avg	Stdev
Overall project satisfaction?	1 = very unsatisfied, 5 = very satisfied	4.45	0.76
How well did the group work together?	1 = terrible, 5 = wonderful	4.47	0.56
Amount of requirements that changed?	1 = everything, 5 = nothing	2.50	1.08
Predictability of requirements changes?	1 = very predictable, 5 = very unpredictable	2.87	0.70
Helpfulness of Redmine?	1 = very unhelpful, 5 = very helpful	3.97	0.94
Usefulness of group workdays in lecture?	1 = very useless, 5 = very useful	4.11	0.98
How burdensome were the reports?	1 = very burdensome, 5 = very unburdensome	3.50	0.98
Customer attentiveness?	1 = very inattentive, 5 = very attentive	3.79	1.19
Customer feedback usefulness?	1 = completely useless, 5 = very useful	3.71	1.23
Satisfied with the customer?	1 = very unsatisfied, 5 = very satisfied	4.11	0.89
Recommend customer again?	1 = never, 5 = completely	3.97	1.10
Mentor attentiveness?	1 = very inattentive, 5 = very attentive	3.34	1.02
Mentor feedback usefulness?	1 = completely useless, 5 = very useful	4.18	0.80
Satisfied with the mentor?	1 = very unsatisfied, 5 = very satisfied	3.89	0.92
Recommend mentor again?	1 = never, 5 = completely	4.16	0.95
Overall satisfaction with project?	1 = very unsatisfied, 5 = very satisfied	4.50	0.56
Choose project again?	1 = definitely not, 5 = definitely	4.03	1.13
Opinion of framework used?	1 = worst thing ever, 5 = greatest thing ever	4.18	0.87

Table 1: End of year survey questions and results ($n = 38$)

The students were required to report their hours. The projects had, on average, 703 hours spent throughout the entire year. The standard deviation was high (153), as might be expected from a group of such disparate projects. We believe the number of hours spent to be much higher, as we know many students did not report all of their hours.

9. CONCLUSIONS

We consider last year's SLP to have been a success. The students were very satisfied with their projects, and the nonprofits were happy with the results. The nonprofits were all presented with systems that fulfilled the agreed-upon requirements. For many of the nonprofits, this will save them a significant amount of time. We have addressed the reasons some projects did not fully succeed; many of those are inclued in the lessons learned section, above.

Based on the number of hours contributed to each project (we had students track their time) we generated an estimate for the monetary contribution of each project. If you assume a $100 salary per hour for a software developer, then the average contribution per project was $69,852 – and there were 7 projects. Thus, there was almost half a million dollars of free software development contributed to the community. Even if you assume a lower per-hour salary, it is still a significant contribution.

The nonprofits for the current year are quite excited about the projects that have been selected. Many of the projects will save the respective nonprofit staff members multiple hours of work per week. Others will save a significant amount of money from not having to hire a professional developer to create such a system for them. The projects will allow those nonprofits to focus their resources – both staff time and financial resources – on other areas to better help the community.

10. FUTURE WORK

This past summer a significant amount of time was spent approaching the nonprofits about potential projects, as well as trying to scope out these projects. As word of this program spreads, we hope to have an application process where the nonprofits will apply for such a project. The goal is to work with the nonprofits to ensure that their requested project meets the necessary requirements (viable within a one-year course, of sufficient complexity, etc.). A committee of faculty, professional software developers (likely the mentors of the projects), and individuals from the nonprofit community will then decide which projects are the best fit for the SLP and the community. We intend to implement this in the spring of 2014 for the 2014–2015 academic year.

11. REFERENCES

[1] CNE. *Center for Nonprofit Excellence*, 2012. `http://www.thecne.org`.

[2] M. A. L. Egan and M. Johnson. Service learning in introductory computer science. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, ITiCSE '10, pages 8–12, New York, NY, USA, 2010. ACM.

[3] L. Layman, L. Williams, and K. Slaten. Note to self: make assignments meaningful. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, SIGCSE '07, pages 459–463, New York, NY, USA, 2007. ACM.

[4] J. Mertz and S. McElfresh. Teaching communication, leadership, and the social context of computing via a consulting course. In *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, pages 77–81, New York, NY, USA, 2010. ACM.

[5] J. A. Stone, B. MacKellar, E. M. Madigan, and J. L. Pearce. Community-based projects for computing majors: opportunities, challenges and best practices. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 85–86, New York, NY, USA, 2012. ACM.

[6] C. Traynor and M. McKenna. Service learning models connecting computer science to the community. *SIGCSE Bull.*, 35(4):43–46, Dec. 2003.