

Data Information Literacy Case Study Directory

Volume 1 *Purdue University*
Number 1 *General Engineering*

Article 1

2015

Electrical and Computer Engineering/ Undergraduates/ Carlson & SappNelson/ Purdue University/ 2012

Jake Carlson
Purdue University, jakecar@umich.edu

Megan R. Sapp Nelson
Purdue University, msn@purdue.edu

Follow this and additional works at: <http://docs.lib.purdue.edu/dilcs>

 Part of the [Electrical and Computer Engineering Commons](#), and the [Library and Information Science Commons](#)

Recommended Citation

Carlson, Jake and Sapp Nelson, Megan R. (2015) "Electrical and Computer Engineering/ Undergraduates/ Carlson & SappNelson/ Purdue University/ 2012," *Data Information Literacy Case Study Directory*: Vol. 1: No. 1, Article 1.
<http://dx.doi.org/10.5703/1288284315477>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

ADDRESSING SOFTWARE CODE AS DATA: An Embedded Librarian Approach

Jake Carlson, University of Michigan

Megan Sapp Nelson, Purdue University

INTRODUCTION

This Data Information Literacy (DIL) team, one of two Purdue University teams in the Institute of Museum and Library Services (IMLS)–funded project, partnered with software design teams involved with Engineering Projects in Community Service (EPICS), a course for undergraduate students from a variety of disciplines. We primarily worked with the graduate teaching assistants (TAs) who graded undergraduate design submissions produced during the design cycle. The software teams created code-based data sets and supporting documentation in a variety of languages and platforms. The creation of code documentation was the primary DIL need of the software teams.

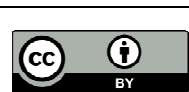
To respond to these needs, the Purdue DIL team developed a rubric that provided guidance for students to create and TAs to evaluate the documentation. Our team created a series of suggested exercises for students that tied specific data management activities to phases of the engineering design cycle used by EPICS (Lima & Oakes, 2006). We then implemented an embedded librarian service within the software teams. We handed out the rubrics and suggested exercises, offered a skill-training session to further enrich the students' knowledge, met with the TAs to help them understand the document, and then served as design reviewers (outside assessors) for the teams.

To assess the intervention, we used the design notebooks created by individual team members to identify instances where the students demonstrated DIL objectives. We created a coding schema that standardized notebook analysis across teams. The assessment concluded that on the individual level, students did not adequately record their coding decisions or articulate the rationale behind these decisions.

While students showed a range in skill level in personal mastery of DIL, widespread weakness was evident in the competencies of data management and organization, data curation and reuse, and data quality and documentation. The core of our program was the integration of librarians within a preexisting, highly structured course. In the future, we plan to focus on implementing a role within the team that is responsible for ensuring that the documentation is of sufficient quality that it can be easily understood and is complete enough to ensure continued development of the project.

ENVIRONMENTAL SCAN OF DATA MANAGEMENT PRACTICES FOR SOFTWARE CODE

Data curators and digital preservation experts are paying more attention to software code as it is not uncommon for code to be an important component of a data set or other electronic object (Matthews, Shaon, Bicarregui, & Jones, 2010). If the data set is to be curated effectively, it logically follows that the

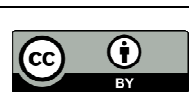


This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

accompanying code must be accounted for in all curation planning and activities. Managing and curating software code as a component of a data set presents several challenges in addition to the ones that would otherwise be encountered in curating data. These challenges include the myriad of components and dependencies of code (such as externally focused documentation, internal documentation, multiple versions of iterative code created, and so forth), the practice of building on or incorporating code developed over time or from multiple authors, and the rapid pace of new technologies that are introduced and adopted by software code writers. Therefore, data sets that include software code may require additional planning and consideration.

Although the literature on the curation of software code as a component of a data set specifically is relatively limited, there is a great deal of literature that touches on the 12 DIL competencies and software code more generally. Data management and organization, and what we referred to in the DIL project as data quality and documentation in particular, have received a significant amount of attention. We focused our environmental scan on a subset of material that appeared most relevant to address the issues faced by EPICS. We also selected a range of materials that touched on each of the 12 competencies in some way. The selected materials in our review included scholarly articles, trade publications, reports, books, and websites to incorporate the perspectives of both academics and professionals in the field.

This environmental scan was helpful in informing our work in several ways. Code developers have a reputation for sharing their work with others as a matter of practice. For example, the ideas of “open source” and “open access” are assumed to be a strong component of the culture of practice of developers, which was largely supported in our literature review (Crowston, Annabi, & Howison, 2003; Hal-loran & Scherlis, 2003). However, despite an ethos and willingness to share code, many developers do not provide the documentation necessary for others to understand or make use of their code easily (Sojer & Henkel, 2010; von Krogh, Spaeth, & Haefliger, 2005). Furthermore, code comments or other descriptions are often absent, or do not reflect the intent of the coder sufficiently, making it difficult if not impossible to understand the decisions made in developing the code (Marcus & Menzies, 2010; Menzies & Di Stefano, 2003). This is despite the availability of resources to assist in the documenting process in software repositories and the availability of tools such as Doxygen (n.d.). Software coding is frequently a collaborative activity, particularly in the workplace, as coders will often be assigned to work on existing code as a part of a team whose membership will change as collaborators transition in and out of a project. Documentation, description, and organization of code are all recognized as important activities for a software group, but they are often activities that are neglected (Lethbridge, Singer, & Forward, 2003). Many researchers in the computer science field present these issues as research questions to solve and suggest technology based solutions to address them (Bettenburg, Adams, Hassan, & Smidt, 2010; Grechanik et al., 2010; Hasan, Stroulia, Barbosa, & Alfifi, 2010). However, these proposed technology-based solutions are often more theoretical than applied in nature by design and therefore of limited practical value.

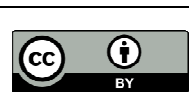


This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The environmental scan led to several other observations and findings that informed our work with EPICS. We noted some related interests within the curation and software communities but found that they used different terminologies in expressing these interests. For example, the idea of “software traceability”—or the practice of recording design decisions including the who, what, where, when, and why and explicitly connecting these decisions to the software for the purposes of quality assurance (Ali, Gueheneuc, & Antoniol, 2011; Bashir & Qadir, 2006)—has commonalities with the data curation idea of “provenance,” or tracking and accounting for actions and decisions made in curating a digital object (Bashir & Qadir, 2006). Traceability is a quality assurance process ensuring that design decisions are readily identified and accounted for over the course of developing the code. Provenance is tracked to ensure the integrity of the existing object and to demonstrate compliance with the policies and practices of the repository. It is the difference between developing something and maintaining it. We also came across a school of thought that advocated for “literate programming” and “human readable code.” The essence of the argument was that rather than creating code to solely be machine readable, developers should create code with the deliberate intent of making it suitable for human reading as well (Knuth, 1984). An offshoot of this idea, “clean code,” was particularly useful in planning our educational programming (Martin, 2008). Finally, the need to preserve software code seems to be catching on in the data curation field, though we did not observe this as much in the software literature, where there seems to be a “technology moves too fast” mentality (Chen, 2001). One particularly useful resource in this area of preservation is the Software Sustainability Institute (<http://www.software.ac.uk/>), which provides services and resources to ensure that software used in research is available and supported beyond its original life span.

METHODOLOGY

Our project partner was Engineering Projects in Community Service (EPICS), a service-learning center at Purdue University (<https://engineering.purdue.edu/EPICS>). EPICS is focused on teaching undergraduates engineering design concepts and skills by working with community service agencies to develop customized engineering solutions that address real-life needs. EPICS brings students from a variety of disciplines across the university and academic years to work together on a common project. Therefore EPICS capitalizes on the diversity of strengths that the participating students bring each semester, but also must manage the gaps in their knowledge and abilities. This is a highly transitory group of students, with project personnel turning over each semester as projects continue till completion. One of the librarians on this project, Megan Sapp Nelson, worked with EPICS on previous projects and had developed a strong understanding of their information needs generally, as well as their working culture. As an advisor to EPICS software teams for 4 years, she was familiar with the highly structured nature of the design course and had previously developed information literacy education interventions to improve the quality of the conceptual design performed in the projects (Sapp Nelson, 2009, 2013). From past experiences, she was aware that students had difficulty managing their software code and documenting their work, which presented problems for all involved, including future students coming into the project, faculty advisors and administrators in EPICS, and the community partners who will make use of the students’ projects.



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The DIL team interviewed four faculty and two graduate students in the spring of 2012 using a modified version of the Data Curation Pro- files Toolkit instrument (available for download at <http://dx.doi.org/10.5703/1288284315510>). To incorporate a broad perspective on managing and curating software code, we interviewed individuals who were affiliated and unaffiliated with EPICS and who came from three disciplines. Table 5.1 shows the affiliations of the interviewees.

TABLE 5.1 Purdue DIL Team Interviewees by Department and Affiliation

DIL Interviewee	Academic Discipline	EPICS Affiliation
Faculty #1	Electrical engineering	Affiliated
Faculty #2	Engineering education	Affiliated
Faculty #3	Computer science	Nonaffiliated
Faculty #4	Computer science	Nonaffiliated
Graduate student #1	Electrical engineering	Nonaffiliated
Graduate student #2	Computer science	Nonaffiliated

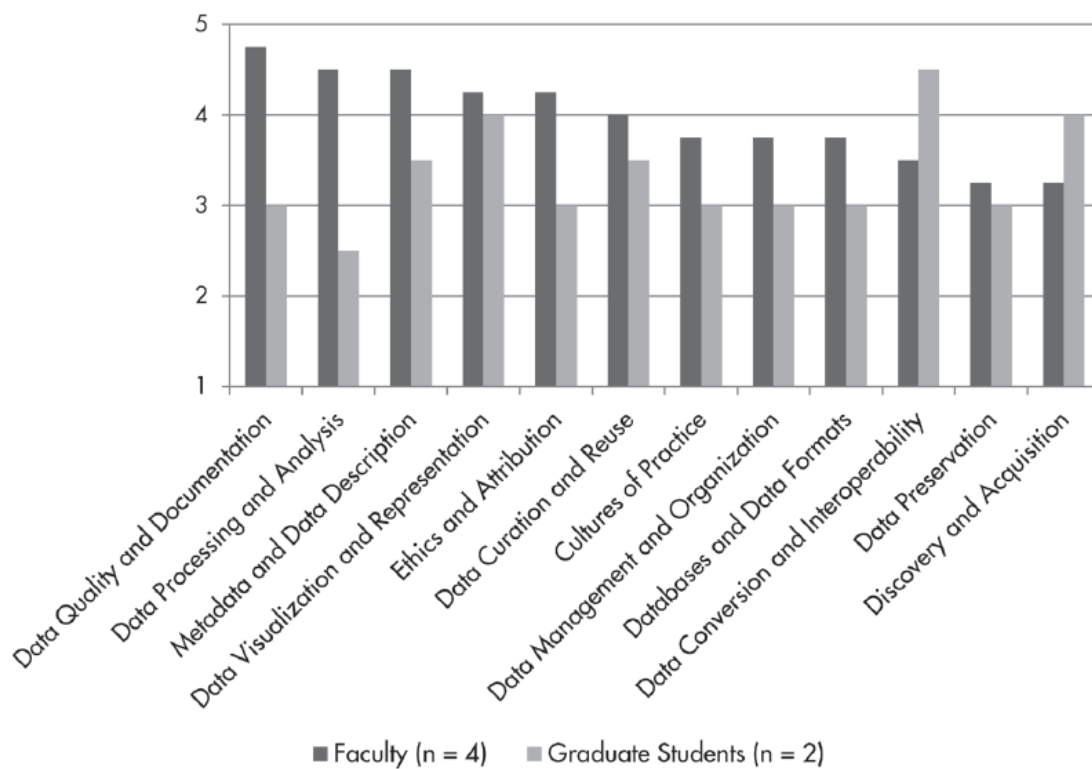


Figure 5.1 The average ratings of importance for each of the 12 data competencies for faculty and students interviewed.

Results of the Needs Assessment

Both the faculty and students rated each of the 12 DIL competencies on a 5-point scale according to how important it was for graduate students to master the competency. The rating results by our six participants are presented in Figure 5.1.

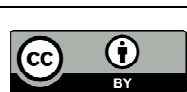
Among the top DIL competencies for the faculty we interviewed were data quality and documentation and metadata and data description. It is interesting to note that faculty rated these two competencies much higher than the graduate students did, demonstrating a disconnect between the attitudes and perceptions of faculty and students in these areas. Furthermore, these two are highly rated within the 12 competencies on average, despite students indicating that they place less importance on them. Faculty recognized data quality and documentation in developing software code as a weakness in students. While students frequently are instructed to document code development, their understanding of what this documentation should consist of and the degree to which quality documentation is necessary are often misunderstood, which leads to high variability in their team's performance and in the quality of the code. Faculty recognized metadata and data description as important. However, while faculty were aware of the need for metadata, they reported that they themselves did not have the understanding or skills to apply metadata nor to teach their students about it.

Conversely, graduate students rated data conversion and interoperability and discovery and acquisition higher in importance than the faculty. For data conversion and interoperability, this is likely due to one faculty member stating that her lab did not engage in converting data, and another stating that this was not a skill that all students needed as long as they had access to someone knowledgeable in this area. Rather, the area of particular interest for both faculty and students within this competency was the prevention of data loss in the conversion process. For the discovery and acquisition competency, the faculty indicated that it may not always be crucial to the research being conducted. For example, their projects were not making extensive reuse of software code. However, the graduate students stated that they will search for existing code that performs similar functions to the code that they were generating, which may explain their rating of this competency as more important than the faculty's. Interestingly, we found that the primary means of locating existing code for the graduate students and faculty we interviewed is a literature search of conference proceedings. A literature search is then followed by a Web search to find the project or author's website where the code may be available.

On the basis of the interviews, our environmental scan, and our knowledge of EPICS, we developed and built the educational intervention around the data quality and documentation and the metadata and data description competencies. Our intended audiences were the graduate student TAs and their undergraduate team members in the EPICS program.

OVERVIEW OF THE EPICS ENVIRONMENT

The EPICS curriculum develops engineering design and professional skills in an environment intended to be a bridge to the students' professional careers. EPICS is a highly structured and intense environment



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.


as students must take on a fair amount of work in new and unfamiliar areas and are held to high standards of professionalism by their instructors.

This environment requires students to take initiative in developing their assigned projects independently but with the knowledge that their instructors will evaluate their work and performance. Consequently, students receive rubrics that will be used for evaluations so that they better understand what is expected of them. Students also learn the design life cycle, a framework for developing and executing their projects (Lima & Oakes, 2006). Students map their work to the stages of the design life cycle as they progress through the course. The work is performed in teams, and within each team students assume particular roles, such as team leader or as primary contact for the project partner (see Table 5.2). EPICS uses a number of different approaches to develop these skills. Typically, at the beginning of the semester, EPICS holds introductory lectures for students that include distribution of the rubrics that will evaluate their performance. Next, students participate in a series of skill sessions to teach them some of the fundamentals they will need to know to be successful, such as programming languages, team building skills, and appropriate use of laboratory resources. All students meet for weekly lab sessions during the semester, where they discuss their progress and the challenges they have encountered while working with their team. As the semester progresses, students present their work in two separate design review sessions, which often include a representative from the project partner organization and professional engineers. There, students receive feedback and suggestions on their work and the quality of their presentations.

TABLE 5.2 Defined Team Roles in the EPICS Curriculum

Role	Responsibility	Faculty, Graduate, or Undergraduate (F/G/U)
Team leader	Team member responsible for overseeing all projects conducted by team in a given semester	U
Project leader/ manager	Team member responsible for overseeing work on a single project for a given semester	U
Project partner liaison	Team member responsible for initiating and maintaining communication with community partner	U
Advisor	Faculty member assigned to oversee the student team for a given semester	
Graduate teaching assistant	Graduate student responsible for providing resources, holding team accountable, and grading	G

In EPICS, students are expected to produce documentation that describes their own work as well as the decisions and actions taken by the team to accompany their coding files. Students organize their data sets using multiple techniques. The primary sources of project-level documentation are the design notebooks or blogs required for completion of the EPICS class. Students store their notebooks in a

	<p>This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.</p>
---	---

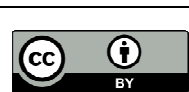
physical location near the lab meeting place or on a server in their digital form. The internal project management documents and the external or user documentation are in a variety of Microsoft Office files and are located on a server, wikis, or Subversion (SVN). Teams manage and store the code itself using SVN. They write code using software languages such as C++ and JavaScript as well as utilizing the Android and Apple mobile platform development tools. Depending upon the team, there may be several software code data sets under development at any given time.

Within the EPICS environment, it is very important to be able to share code both within a team and outside of it. As projects typically span multiple semesters, students will transition in and out of the team over the life of a project. As such, a need within EPICS is that the resulting code and code structure be readily apparent, logical, and “human readable” to facilitate the transition between developers on each project. Another consideration is that the software code has real-world application outside of the educational realm. The code is designed for practical use by nonprofit agencies in the local community. It is therefore very important that the code be designed and delivered in ways that support its ongoing use and maintenance over time. More information about EPICS can be found on its website (<https://engineering.purdue.edu/EPICS>).

The challenge for the DIL team involved supporting the development of useful software code products, which was a complex endeavor made more complicated by the high rate of turnover among team members between semesters. TAs are asked to hold their undergraduate student team members accountable for the quality of their code during the grading process. However, it was evident from the interviews that the TAs did not have the experience, comfort level, or tools to grade the quality of the code and the documentation that the students were submitting, and ultimately they had difficulty holding the team members accountable.

EPICS as a whole did not have a cohesive, clearly articulated culture of practice regarding the management and documentation of code. Some teams agreed to naming conventions for files and variables or developed other “local” standards, but this was left up to the individual teams to decide. Generally, the code writers looked to more experienced teammates to provide them with standards, rather than developing standards among the group by consensus. A few faculty advisors provided expectations for code documentation, but it was not a standard across EPICS and happened infrequently.

A variety of development tools were used as needed by individual teams that supported creating documentation for code, such as JavaDocs (<http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html>) and Yii (<http://www.yiiframework.com/>). TAs supervised more than one team, which meant that the TAs had to familiarize themselves with the tools that each team was using. On some of the teams new students went through multiple weeks of training to teach them how to use the tools as well as introductory coding skills. TAs provided guidance during this process and one-on-one instruction for student coders who were having difficulty.



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

Faculty advisors generally agreed that the level of oversight for student coding projects was insufficient. The TAs indicated that part of the difficulty in providing oversight was a subjective measure of quality for the coding. Although EPICS faculty and TAs raised documentation, organization, and transferability of the software code as serious issues, they had not yet developed supporting materials or strong cultures of practice in these areas within EPICS. Therefore the DIL team saw an opportunity to support the work of the TAs, who in turn supported the education of undergraduates in the EPICS program, through developing resources and providing a framework for good software code documentation practices.


TABLE 5.3 - Learning Objectives for Students and Teaching Assistants in EPICS

Target Audience	Learning Objectives
Undergraduate students who are a part of software development EPICS teams will:	<p>Recognize that documentation and description are integral components of developing software code (and are not simply “busy work”) in order to hold oneself and team members accountable for producing quality documentation and description in a timely manner</p> <p>Document own code and methods in developing the code in ways that enable the reproduction of work by others in order to ensure the smooth transfer of work to other students and the EPICS project partner</p> <p>Create and communicate standard operating procedures for managing, organizing, and documenting code and project work within the team in order to develop consistent practice and to facilitate clear communication amongst team members</p>
Teaching assistants who lead software development EPICS teams will:	<p>Identify characteristics of well-written software documentation in order to recognize well-written project and software documentation</p> <p>Evaluate project and software documentation in order to identify both positive and negative data practices</p> <p>Critique project and software documentation in order to assess quality and assign grades</p>

AN EMBEDDED LIBRARIAN APPROACH TO ADDRESSING DATA INFORMATION LITERACY NEEDS

The DIL team developed goals and learning objectives for educational programs based on the results of the interviews, environmental scans, and previous knowledge of EPICS. They had three overarching goals:

1. To raise the students’ awareness of the need to generate quality documentation and description of the software code they generated
2. To provide students and graduate TAs with the knowledge and tools to generate quality documentation and description for software code

	<p>This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.</p>
---	---

3. To develop a shared cultural practice in EPICS based on disciplinary values in data management issues, particularly issues in quality, documentation, and the description of data and software code

Table 5.3 lists the specific learning objectives for the two target audiences.

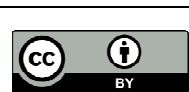
Given the structured nature of EPICS and the intensity of the work, the DIL team found that the students had little time for “additional” learning activities or events. So we decided to take an “embedded librarian” approach to developing and delivering a DIL educational program that connected with the EPICS structure and culture. Embedded librarianship can be defined as the process of presenting information literacy content as a part of course curricula in ways that are directly relevant to student outcomes for the course (Schulte, 2012). Embedded librarianship is a particularly promising method for implementing information literacy instruction due to the presentation of information literacy competencies in an immediately relevant manner (Tumbleson & Burke, 2010). Given the project-based nature of the course, an embedded librarianship approach appeared to best integrate with the course design and content that already existed within the EPICS program.

To implement our embedded librarian approach, in the fall of 2012 we focused on three groups within EPICS. Each of these groups had at least one faculty advisor, a graduate student TA, and multiple teams of students that each worked on a particular project. Our approach for implementing our educational programming was to forge connections with the faculty advisors, graduate TAs, and students in EPICS by taking advantage of built-in opportunities to interact with each group. This included

- developing an evaluation rubric for TAs to apply to student work;
- offering a skills-based session on documenting code and project work;
- attending lab sessions and observing team meetings;
- participating as reviewers in the students’ design review sessions.

To create this educational program, we first returned to the literature review, particularly the sources that described criteria for developing “clean code,” to identify relevant best practices and documentation guidance for software developers. Next, using the existing rubrics developed by EPICS as a guide, we crafted two rubrics (Appendix A to this chapter) that the graduate TAs could use to evaluate both the code and the documentation created by their students. We also distributed a one-page document (Appendix B to this chapter) to team leaders that explained the expectations for quality code and described why documentation of code is important. Finally, we shared our work with the TAs and made some adjustments based on their feedback. Table 5.4 shows the full schedule.

We held the skills session on documenting and organizing code during the third week of the semester. The focus was on helping the team leaders in EPICS recognize what constituted quality, professional practice in documenting and organizing code, and the need for students to internalize these practices. The session comprised three modules (see the complete lesson plan in Appendix C to this chapter). In the first module we presented quotes from articles written by several prominent coders that described



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

the attributes of “clean code.” We then distributed three examples of code that had been generated by previous EPICS teams. We asked the class to identify the strengths and weaknesses of the code from the perspective of documentation and organization. We closed this module with a discussion of what constitutes good code versus poor code. In the next module we discussed why writing well-documented and well-organized code matters. We emphasized that writing software code is inherently a collaborative activity as the majority of code will be used by others, both as a product and also as something edited and maintained by other coders (future EPICS students in this case). We then introduced a coding skills inventory (see Table C.1 in Appendix C to this chapter), a list of 12 skills to facilitate good coding habits in EPICS teams. In the last module, the team leaders picked one of the skills on the coding skills inventory list that they saw as a high priority for their team and designed a short learning activity that would address this skill. We provided the team leaders with activities that could support such an intervention (see the list in Appendix D to this chapter). We recognized that the teams were at different stages in the software development process, so we mapped our list of activities to the stages of the design life cycle to facilitate this process. Finally, each team leader shared a selected skill and activity with the group and de- fined the measure of success for the activity.

Unfortunately the skills session was voluntary and there was a poor turnout. While all team leaders and project leaders were invited, only five students attended from four teams. We found that this introduction to DIL skills was not pervasive enough to introduce and instill a foundation of good practice.

TABLE 5.4 - Embedded Librarian Engagement Activities

Semester Timeslot	Activity	Description
Week 2	Introduction	Initial visit to the EPICS weekly lab session to introduce the DIL team and distribute rubric materials to all students
Week 3	Voluntary skills session on documenting and organizing code	This session was offered to team leaders in EPICS and covered the following: Module 1—What is good coding? Module 2—Why is it important? Module 3—How to foster good coding practices in your team
Weeks 4–6	Embedded librarianship	Observations and consultations in weekly lab sessions
Week 7	Design review #1	First round of feedback and suggestions for student work in documenting their code and their projects
Weeks 8–13	Embedded librarianship	Observations and consultations in weekly lab sessions
Week 14	Design review #2	Second round of feedback and suggestions for student work in documenting their code and their projects
Post-semester	Assessment	Collected and reviewed student lab notebooks



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

As the semester progressed we made frequent visits to the EPICS labs. Early in the semester we attended a lab for each of the three teams we were working with and introduced ourselves to the students. We distributed the documentation rubric that we had developed. Subsequently, we each attended multiple lab sessions for each of the three groups over the course of the semester. These interactions gave us the opportunity to observe how students were developing their work and to interact with them (though in a limited fashion as lab sessions covered many aspects not related to the DIL project). We also attended both of the design reviews (7 weeks and 14 weeks into the semester) and were able to provide some suggestions for their work in documenting their code and their projects.

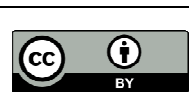
Our approach in assessing this work has been twofold. First, we met individually with two of the three TAs for the teams (the third was unavailable) and two of the faculty advisors at the end of the fall 2012 semester. We asked about any changes in student behavior they observed, changes in their perceptions of these topics, and possible next steps for our work with EPICS. Although the feedback we received was generally positive, no one reported a substantial change in student activities in writing code and documenting their work. They encouraged the DIL team to keep working with EPICS, and as a result of these conversations, developed some ideas for the future as described in the “Discussion” section. Second, we reviewed the lab notebooks that students in one of the groups we had worked with had written during the fall semester. The DIL team developed a coding schema to evaluate student knowledge and skills in documenting their work effectively. This analysis will enable us to better pinpoint areas of need and will inform our work in developing more targeted responses.

DISCUSSION

The opportunity to embed within a highly structured, multiple section class provided this Purdue DIL team a broad range of insights for actionable next steps, future research, and recommendations to the EPICS leadership team.

First, we identified that the team leader and project leader roles are key to the dissemination of good data management planning and practice within any given team. We identified this early through interviews and attempted to address this via a one-shot skill session aimed at the student project and team leaders. Given the low level of turnout and lack of observed knowledge/skill transfer from the session, we needed to develop a more embedded approach to data management skills building.

Another differentiating aspect of the EPICS environment is the assignment of specific roles to students within their groups. Teams in EPICS select their project and team leaders early in the semester, along with more specific roles such as the webmasters, project partner liaisons, and financial officers, among others. Despite the near ubiquity of teams encountering issues with the documentation done by previous students, teams do not acknowledge this issue in their meetings or do much to address it formally. A defined role for a student member of a team might ensure that code documentation and description of the project were carried out efficiently and in ways that ensured a smooth transition from semester to semester, as well as from EPICS to the community agency when the project is done. The current approach of having students share the responsibility of documentation and description instead



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

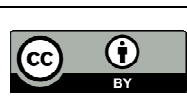
of designating a member of the team to have direct ownership of these tasks is a major cause of the low-quality documentation and difficulties in the transfer of work.

Therefore, the DIL team proposed a pilot project for the fall of 2013 to define and implement a project archivist role within selected EPICS teams. The purpose is to integrate fully the oversight of documentation formally within the team structure by creating a specific team role. We envision the project archivist's role as taking a big picture approach toward capturing the description and documentation of the project, including the design constraints, decision-making processes, and design implementations for each team. As a result, the EPICS teams might see smoother transitions of the project to future team members, graduate teaching assistants, faculty advisors, EPICS administrators and project partners. We will be working with a continuing lecturer and an EPICS advisor to further define, implement, and assess the impact of the project archivist role.

Second, while the rubrics for evaluating software code and documentation that we developed are a good start, there is a need for further curricular development to integrate the rubric into the EPICS workflow for the semester. A high priority will be to address the individual and team documentation templates used by EPICS. Currently, these templates do not highlight the need for excellent coding practices and data management. Working with the EPICS administrative team, we hope to create a template or other workflow that highlights the need for well-designed and well-written code while providing a structure for individual and team-level accountability. These resources will support the TA's role as a mentor to EPICS students, using a train-the-trainer approach.

Another need that the DIL team identified was a central reference solution that enables students (both undergraduate and graduate) to learn needed data skills at their point of need, while working either independently or in a laboratory setting. We feel that a library of short videos (perhaps hosted on a YouTube channel) that covers software and data management topics would be highly useful to EPICS. The EPICS curriculum is built around the idea of working independently to write code that is then brought back to the group for further development. It is important that students have instruction on clean coding, creating excellent documentation, and project management planning that is available to them outside of class. Similarly, graduate students frequently work independently, submitting code to their supervisor for comment and review. A YouTube library would create a ready reference for those needs that arise while the students are practicing or expanding their skill sets.

Finally, we noted that the depth and quality of project documentation and reflection captured in the team members' lab notebooks varied widely. The highest order of learning skills according to Bloom's taxonomy (Bloom, 1956)—evaluation and analysis—were not often present within the EPICS notebooks, even as the students were engaging in a creative process. Evaluation and analysis are at the heart of excellent data management skills; by looking at the long-term life span of the project, students identified the immediate worth of clean code not only for themselves but also for future EPICS team members, project partners, clients, and users. Working with the EPICS administrators, we hope to emphasize the reflective practice of code writing, particularly for software and hardware engineering disciplines.



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

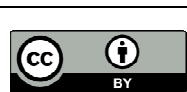
CONCLUSION

This approach toward developing and implementing a DIL educational program was to embed into the structure and environment of EPICS. Embedded librarianship was a natural choice given the highly structured nature of the EPICS program and engineering disciplines. This approach allowed us to reach a relatively large number of students (40 approximately) in ways that aligned with their current practices. However, employing an embedded librarian approach in our program took a great deal of planning and investment for the DIL team to set up and carry out.

Several interrelated factors should be addressed in this type of DIL model. First, the embedded librarian approach requires that librarians build solid relationships with the people running the program. When a librarian is embedded in a course, this may include just the faculty instructor and his or her teaching assistant. We decided to partner with a service-learning center and to focus our efforts on three groups and their graduate student TAs overseeing the work of multiple teams of students. This structure required us to build connections with the faculty advisors, the graduate student TAs, the EPICS administration, the student team leaders, and others. Sapp Nelson's prior experience aided our relationships in working with EPICS, as did Carlson's previous interactions with one of the faculty advisors. Nevertheless, our approach still required multiple meetings to introduce ourselves, explain what we were trying to do, and establish contact with a great number of people. We recommend that librarians who wish to launch a DIL program plan to cultivate and maintain relationships as a part of their program development.

Second, we worked hard to align our efforts to fit into the structure of our partner. EPICS has a very structured way of doing things that did not allow for a great deal of deviation. Therefore, we had to identify these structures early on and then determine how best to integrate ourselves to reach students in meaningful ways. We took advantage of opportunities to reach students, such as holding a voluntary skill session early in the semester and attending design reviews at the midpoint and end of the semester. However, we also had to create additional ways of connecting with students within the EPICS structure. Our approach was to align our instruction and interactions as best we could with current practices. We did this by creating a rubric for evaluating student documentation and organization practices and making ourselves available during some lab sessions.

Third, the embedded librarian approach required a fairly significant time commitment. In addition to the time that we invested in identifying which of the DIL competencies to address and in developing the knowledge to design an educational program to respond, the DIL team put in many hours attending lab sessions and design reviews, offering the skill session, developing resources, and meeting with faculty advisors and TAs affiliated with EPICS. We believe that the in-person contact was worth the effort as it definitely helped make an impact, forge relationships, and better understand the EPICS environment. However, it was occasionally difficult to find the time to devote to making these personal appearances given our other responsibilities and because we followed EPICS's schedule rather than our own. The time commitment continues as we review the content of team lab notebooks to better determine the impact the DIL program had on students and to observe where their DIL competencies strengths and



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

weaknesses lie. Here too, we believe that the time commitment in assessing student work will pay off as we continue to develop our partnership with EPICS.

Beyond the lessons learned from developing the program itself, we gained a better understanding of the 12 DIL competencies from the interviews. We decided to focus on only 2 of the 12 competencies for our work with EPICS on the basis of its needs and our ability to respond to those needs. However, the needs expressed were many and may provide additional opportunities for follow up. In particular both the faculty and the students we interviewed indicated that competency with data visualization and representation was important. In addition to the breadth of needs expressed in the interviews, we observed wide variations in baseline skills of students working with EPICS. For this project, we deliberately kept the definitions of the competencies loose, as we wanted interviewees to express their opinions and perspectives on the competencies with little direction or interference from us. For our work with EPICS on data quality and documentation, it was clear that its success is very much specifically oriented on a particular skill in that competency: “Documents data sufficiently enough to enable the reproduction of the research results and the data by others.” However, we needed to define what this statement really meant for EPICS and how it was (or was not) understood by the students, TAs, faculty advisors, and EPICS administration to be able to respond effectively. Learning the context and gaining an understanding of the setting were as important to our program as defining our terms. This was very much an iterative process.

NOTE

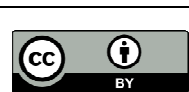
This case study is available online at <http://dx.doi.org/10.5703/1288284315477>.

REFERENCES

Ali, N., Gueheneuc, Y. G., & Antoniol, G. (2011). Trust-based requirements traceability. In Program Comprehension (ICPC), 2011 IEEE 19th International Conference on (pp. 111–120) [IEEE Xplore Digital Library version]. <http://dx.doi.org/10.1109/ICPC.2011.42>

Bashir, M. F., & Qadir, M. A. (2006). Traceability techniques: A critical study. In Multitopic Conference, 2006. INMIC '06. IEEE (pp. 265–268) [IEEE Xplore Digital Library version]. <http://dx.doi.org/10.1109/INMIC.2006.358175>

Bettenburg, N., Adams, B., Hassan, A. E., & Smidt, M. (2011). A lightweight approach to uncover technical artifacts in unstructured data. In Program Comprehension (ICPC), 2011 IEEE 19th International Conference on (pp. 185–188) [IEEE Xplore Digital Library version]. <http://dx.doi.org/10.1109/ICPC.2011.36>



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

Bloom, B. S. (1956). *Taxonomy of educational objectives, Handbook I: The cognitive domain*. New York: David McKay Co. Inc.

Chen, S. (2001). The paradox of digital preservation. *Computer*, 34(3), 24–28.
<http://dx.doi.org/10.1109/2.910890>

Crowston, K., Annabi, H., & Howison, J. (2003). Defining open source software project success. In *ICIS 2003 Proceedings (Paper 28; pp. 327–340)*. Retrieved from AIS Electronic Library: <http://aisel.aisnet.org/icis2003/28/>

Doxygen. (n.d.). Generate documentation from source code. Retrieved from <http://www.stack.nl/~dimitri/doxygen/index.html>

Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., & Cumby, C. (2010). Exemplar: EXEcutable exaMPles ARchive. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Vol. 2 (pp. 259–262)* [ACM Digital Library version].
<http://dx.doi.org/10.1145/1810295.1810347>

Halloran, T. J., & Scherlis, W. L. (2002). High quality and open source software practices. Paper presented at Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering, Orlando, FL. Retrieved from http://flosshub.org/system/files/Halloran_Scherlis.pdf

Hasan, M., Stroulia, E., Barbosa, D., & Alafi, M. (2010). Analyzing natural-language artifacts of the software process. In *Software Maintenance (ICSM), 2010 IEEE International Conference on (pp. 1–5)* [IEEE Xplore Digital Library version]. <http://dx.doi.org/10.1109/ICSM.2010.5609680>

Knuth, D. E. (1984). Literate programming. *Computer Journal*, 27(2), 97–111.
<http://dx.doi.org/10.1093/comjnl/27.2.97>

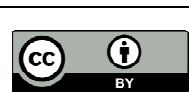
Lethbridge, T. C., Singer, J., & Forward, A. (2003). How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6), 35–39. <http://dx.doi.org/10.1109/MS.2003.1241364>

Lima, M., & Oakes, W. (2006). *Service-learning: Engineering in your community (1st ed.)*. Okemos, MI: Great Lakes Press.

Marcus, A., & Menzies, T. (2010). Software is data too. In *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research (pp. 229–231)* [ACM Digital Library version].
<http://dx.doi.org/10.1145/1882362.1882410>

Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall.

Matthews, B., Shaon, A., Bicarregui, J., & Jones, C. (2010). A framework for software preservation. *The International Journal of Digital Curation*, 5(1), 91–105. <http://dx.doi.org/10.2218/ijdc.v5i1.145>



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

Menzies, T., & Di Stefano, J. S. (2003). More success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, 29(5), 474–477. <http://dx.doi.org/10.1109/TSE.2003.1199076>

Sapp Nelson, M. (2009). Teaching interview skills to undergraduate engineers: An emerging area of library instruction. *Issues in Science & Technology Librarianship*, (58).
<http://dx.doi.org/10.5062/F4ZK5DMK>

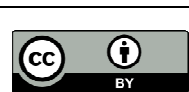
Sapp Nelson, M. (2013). Find the real need: Understanding the task. In M. Fosmire & D. Radcliffe (Eds.), *Integrating information into engineering design* (pp. 87–99). West Lafayette, IN: Purdue University.

Schulte, S. J. (2012). Embedded academic librarianship: A review of the literature. *Evidence Based Library & Information Practice*, 7(4), 122–138.

Sojer, M., & Henkel, J. (2010). Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Economic Policy*, 11(12), 868–901.

Tumbleson, B. E., & Burke, J. J. (2010). Embedded librarianship is job one: Building on instructional synergies. *Public Services Quarterly*, 6(2–3), 225–236. <http://dx.doi.org/10.1080/15228959.2010.497457>

Von Krogh, G., Spaeth, S., & Haefliger, S. (2005). Knowledge reuse in open source software: An exploratory study of 15 open source projects. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)—Track 7 (Vol 7, p. 198b)* [CS Digital Library version].
<http://dx.doi.org/10.1109/HICSS.2005.378>



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.