Collaborating with Students to Produce High-Quality Production Software

Sean Murthy, Andrew Figueroa, Steven Rollo

murthys@wcsu.edu, figueroa039@connect.wcsu.edu, rollo003@connect.wcsu.edu

Abstract

Agile and DevOps processes are generally considered beneficial to collaboratively producing high-quality software. Thus, employers currently seek people with skills in these two areas, even for entry-level positions which new Computer Science (CS) graduates often fill. However, undergraduate CS programs generally only provide opportunities for students to practice core CS concepts in simple classroom exercises and projects, instead of providing immersive experiences. Industry internships do expose students to real-life development and maintenance (often with emphasis on maintenance), but interns rarely experience the entire product life cycle; many do not even get to see the "big picture".

In this paper, we share the experience of a hands-on co-curricular lab where students learn many practical aspects of collaboratively building high-quality production software alongside faculty members. The important bit is that the process of learning and practicing is itself agile-like, in the sense that students are incrementally taught concepts and given opportunities to realize hands-on why a certain process step is necessary and what some of the alternatives might be.

The lab was founded by a faculty member and has thus far engaged 12 CS undergraduate students including sophomores, juniors, and seniors. The faculty member generally moderates discussions and introduces new concepts, but no distinction is made among any team member based on their standing (faculty or student). The lab is not part of any coursework, and all participation is voluntary. The lab has thus far developed two products, but this paper focuses on just one product—called *ClassDB*—and provides the perspectives of the faculty member and two student collaborators.

The following is a summary of the lab's progress in ClassDB: Development began in Summer 2017 with pre-production versions released throughout summer, and the first production version released by the term's end. 20 students used the software during Fall 2017, and 20 more used it in spring after a major update in winter. The software is rolled out to about 50 students this fall and is being prepared for use by about 4,000 students across the university system. All source code, tests, issues, and documentation are available in a public GitHub repository at http://bit.ly/ClassDBRepo.

Biography

Dr. Sean Murthy is a member of the CS faculty at Western Connecticut State University (WCSU) as well as the founder and director of the Data Science & Systems Lab (DASSL, read *dazzle*). He has extensive experience in software engineering and has developed several commercial software products including a few for Fortune 100 companies. Andrew Figueroa is a CS undergraduate student at WCSU presently in his senior year. Steven Rollo graduated from WCSU in Spring 2018 with a BS degree in CS and is now employed at Fiber Mountain. Prior to graduation, Rollo completed two internships in software engineering and systems integration at Seagate Technology. Murthy, Figueroa, and Rollo are all members of DASSL where they co-develop ClassDB, Gradebook, and other products.

1 Introduction

In the past few years, the software-engineering industry has seen rapid and extensive changes in tools, approaches, and more importantly mindset. The changes are due to an explosion in the number of tools that boost accessibility, productivity, and software quality. They are also due to improved access to reliable information on technology, tools, and processes, as well as due to wider adoption of agile and DevOps processes in both engineering and management.

Due to these changes, employers are naturally seeking people with skills in newer tools and approaches and expect employees to have or develop an "Agile and DevOps mindset", even in entry-level positions which new Computer Science (CS) graduates typically fill. However, universities have yet to change their programs to meet employers' needs, largely because they focus on teaching core CS concepts such as programming, algorithms and architecture. In fact, a typical undergraduate program requires students to complete just one course in software engineering. Some programs offer courses on software quality, but only as an elective. CS 320 at Whitworth University is one such elective course (Whitworth University 2018).

A common limitation of most undergraduate CS courses is that they can only afford to have students practice simple exercises and projects, instead of providing an immersive experience. Students who obtain industry internships do get exposure to real-life development and maintenance (often with emphasis on maintenance), but they rarely experience the entire product life cycle. In fact, many interns report they did not get to see the “big picture”.

We believe the typical undergraduate CS program is structurally unable (at least presently in the US) to prepare new graduates to fully contribute to products and processes in this "agile era". We fear that the new graduates' inability to fully and readily participate results in delayed hiring, and the delay can create a vicious cycle where the graduates fall behind more as time passes.

Our solution to this problem is a *hands-on co-curricular lab* to introduce undergraduates to practical aspects of building software, including an introduction to a lightweight “Agile and DevOps process”. The emphasis is on faculty and students collaborating to build, deploy, document, and maintain high-quality production software. (See Section 2.2 for a definition of terms.)

The solution was conceived by, and is led by, a CS faculty member (the first author of this paper) as part of the Data Science & Systems Lab (DASSL, read *dazzle*) (DASSL 2017b) at Western Connecticut State University (WCSU). The lab runs throughout the year, with special sessions in summer and winter.

Since its founding in Spring 2017, DASSL has engaged 12 undergraduates, including the second and third authors of this paper. It has developed two products, including *ClassDB* (DASSL 2018a) which is now in production for about 50 students, with plans to soon open it to about 4,000 students across four universities and 12 community colleges in the Connecticut State Colleges and Universities system. Also, including this paper, DASSL has thus far produced four peer-reviewed publications (Murthy 2018b, 2018a; Murthy, Figueroa, and Rollo 2018), two of which have student co-authors.

More importantly, each participating student has an online portfolio that readily demonstrates their accomplishments to prospective employers. Indeed, DASSL alumni report that their lab participation has given them a competitive edge and that employers have appreciated their familiarity with modern software-engineering processes and toolchains. (Section 6 provides some examples.)

In this paper, we share our experience with faculty-student collaboration to build and maintain high-quality production software at DASSL. We mainly use ClassDB to provide examples and share our perspectives as significant contributors to DASSL.

The rest of this paper is organized as follows: Section 2 introduces ClassDB and DASSL as well as some terms. Section 3 outlines the educational and engineering processes used at DASSL. Section 4 highlights

the “team culture”. Section 5 describes DASSL’s effort specifically in two areas to maintain high quality. Section 6 provides personal perspectives of each author, and Section 7 summarizes the paper.

2 Background

This section provides some background about the ClassDB application and DASSL, and also introduces some terms. The information about ClassDB is limited to the needs of this paper, but details are available elsewhere (DASSL 2018a; Murthy, Figueroa, and Rollo 2018). Also, later sections of this paper add details.

2.1 ClassDB

Instructors can use ClassDB in teaching courses where students interact with databases. It can be used in both introductory courses where students learn to write queries, and in upper-level courses where students perform analytics and other advanced data-management activities. In addition, ClassDB itself provides many case studies for data management and software engineering courses, as evidenced in WCSU courses CS205 and CS305 (WCSU 2013).

Instructors use ClassDB to provide database sandboxes to students and student teams. Each student has full control of their own sandbox. Likewise, each student team has its own sandbox and all team members have full access to their team’s sandbox. No student is able to read another student’s sandbox, but an instructor can read any sandbox. Instructors can also consult the detailed activity logs ClassDB maintains to review student progress and analyze the logs to provide customized feedback to students.

Task	API call
Add ‘edwall’ as a student	`SELECT ClassDB.createStudent('edwall', 'Ed Wall');`
Create team ‘dragon’	`SELECT ClassDB.createTeam('dragon');`
Add ‘edwall’ to team ‘dragon’	`SELECT ClassDB.addToTeam('edwall', 'dragon');`
List currently registered students	`SELECT UserName FROM ClassDB.Student;`
List activity summary for ‘edwall’	`SELECT * FROM ClassDB.getUserActivitySummary('edwall');`
List my DDL activity (any user)	`SELECT * FROM public.MyDDLActivity;`

Table 1: Example calls to ClassDB API

ClassDB runs entirely within a server instance of a database management system (DBMS). Unobtrusive by design, its existence is ordinarily not apparent to any DBMS user, including instructors, but especially to students. It includes no user-interface elements (UI) and is usable through its application programming interface (API) from any pre-existing DBMS client application. The ClassDB API includes about 100 functions and views, a majority of which are accessible to instructors and a few are accessible to students. Table 1 shows some example calls to the ClassDB API.

ClassDB is implemented in PostgreSQL (“Postgres”) using SQL with procedural extensions (PostgreSQL Development Group 2017). The current version of ClassDB—Version 2.2—is comprised of 2,788 lines of code (LOC) indicating that ClassDB is not a small application, given it is implemented in a declarative language: 2,788 LOC in SQL translates to between 11,152 LOC and 27,880 LOC in the C programming language according to Caper Jones’s Programming Language Tables (Wallshein 2010; Jones 2007).

The core of ClassDB is built using the following DBMS concepts: schemas (each sandbox is a schema), role-based access control, triggers, and server logs. The use of role-based access control (Ferraiolo and Kuhn 1992) is particularly important to simplify the system implementation but still provide the safeguards necessary to maintain both data privacy (no student can see another student’s data) and data integrity (not even the instructor can alter student data). Contrary to what the small LOC count might suggest, the use and careful choreographing of these concepts makes ClassDB a relatively complex system.

2.2 DASSL

DASSL is a research group with broad focus on data science and data-intensive systems, including the creation and maintenance of software applications in these areas. Much of the work at DASSL is in collaboration with CS undergraduates. (WCSU does not have a graduate CS program.)

Membership is open to all WCSU faculty and students. Students wishing to join are generally expected to have completed CS205 (an introductory data-management course), and most students who have completed CS205 will likely also have completed CS140 and CS170 (WCSU 2013) which introduce Java and C++ respectively. In this scheme, students are able to join DASSL after their 3rd semester, but due to a variety of reasons, students are more likely to join after their 4th semester. Some students have even joined DASSL in their senior year.

The first author of this paper is the *lab director* and is involved in all lab activity. The other two authors are charter student members of DASSL and have been involved in most DASSL activities to date.

Participation in DASSL is *not* part of any coursework, and all participation is voluntary. Students do not pay any fee or receive academic credit, and the faculty member does not receive any pay. When funds exist, some students receive a small stipend if their role includes helping the lab director run DASSL.

DASSL builds non-trivial applications that address real needs of real end users, with emphasis on end-to-end engineering including analysis, design, development, testing, documentation, deployment, and maintenance. Most DASSL products are free and open source, and are generally distributed under a Creative Commons Attribution-NonCommercial-ShareAlike license (Creative Commons 2013). As a rule, all work products are stored in GitHub repositories (DASSL 2017a), with many repositories being public.

2.3 Quality orientation

We now introduce key notions of quality that drive software development at DASSL. Specifically, we introduce the terms in this paper's title: "collaborating with students to produce high-quality production software".

By "collaborating with students", we mean both faculty members and students working as a team using tools and processes comparable to those used in professional software development. Specifically, both faculty and student members are expected to participate in all software-development activities and subject their work to the same quality-assurance processes.

By "production software", we mean:

- Software meant for use by people other than its developers, and actively used by others beyond the semester or year in which the software is initially released.
- The development process has clear activities of design, implementation, testing, documentation, release, deployment, and maintenance.
- Software is developed and maintained with a process and fervor comparable to what a company in the business of producing such software might use and possess. This criterion is particularly important because one of DASSL's goals is to introduce undergraduates to practical aspects of building software.

By "quality software", we mean software that is useful, usable, and maintainable. We omit discussing the usefulness aspect of ClassDB, because that is not this paper's focus. By usability, we mean ease of deployment, operation, and maintenance. By "maintainable", we mean low effort to find and fix issues as well as to make enhancements.

By “high-quality software”, we mean software that has few issues that prevent it from being useful, usable, and maintainable. At this point, we have not defined quantitative limits for the number, nature, and severity of issues that define “high quality”, but we are in the process of developing a simple model.

3 Processes

In this section, we outline the educational and engineering processes at DASSL.

3.1 Educational

Summer and winter sessions: In addition to regularly scheduled meetings throughout the academic year, DASSL holds special sessions during summer and winter breaks. Students state their intent to join these sessions 1-2 months prior and discuss previous experience, interests, and availability with the lab director.

DASSL has thus far held three special sessions:

- Summer DASSL 2017 lasted six weeks and engaged eight students. It entailed 6-hour in-person meetings four times a week for six weeks, for a total of 144 hours (the equivalent of three university courses).
- Winter DASSL 2018 ran for two weeks and focused on developing ClassDB 2.0. It involved two returning students and two new students.
- Summer DASSL 2018 was 10 weeks long with three returning students and two new students. The main focus was to create ClassDB 2.1 and 2.2 as well as work on conference publications and presentations.

Meetings: DASSL holds occasional meetings to keep track of progress, share ideas, and provide an open environment to discuss project concerns. A majority of the meetings are online, which is particularly helpful during summer and winter. During the academic year (fall and spring terms), DASSL meets in-person once a month. These meetings discuss topics relevant to DASSL and also discuss issues in DASSL products.

DASSL Day: DASSL students publicly present their work once a semester. In addition to helping students develop skills in technical and scholarly communication, this event serves as a recruitment tool for new students, and a means of informing university administrators of the lab’s progress and contributions.

Agile-like learning: Students are introduced to key concepts and tools in an agile-like manner that emphasizes iterative learning through practical scenarios rather than initial mastery of the theory. Because there are few concrete requirements to join DASSL, none involving prior knowledge of specific tools, an agile learning process is essential to reduce onboarding time and allow students to start contributing early.

3.2 Engineering

Milestone-driven development: Prior to starting development on a project or a product’s release, an informal list of desired topics or features is created as a wiki page (DASSL 2018e). The list is discussed, refined, and expanded (with links to additional wiki pages if necessary). Once consensus develops, the list becomes a “to do” list for a new “milestone” and is assigned a completion date and the resulting product version number. These milestones function much like “sprints” in a traditional agile process. The wiki page for ClassDB Milestone 2 (DASSL 2018c) illustrates our approach.

Issue-driven development: Issues are logged and managed in GitHub Issues (DASSL 2018b). Each issue is classified by *nature* (wrong, missing, extra) and *priority* (low, medium, high). Where appropriate, issues are grouped into "epics". (ZenHub 2018b, 2018a). Section 5.2 details our issue-management process.

Developers self-assign issues to work on and generally choose the next pending issue with the highest priority. However, students new to DASSL tend to work on "easy to address" issues such as those needing changes to a single region in a single file. Regardless of issue classification, all changes committed to the repository are tagged with issues so that both issues and changes can be traced to each other.

Version control: We require every artifact—both code and non-code—to be under version control. To further ensure that work-in-progress does not interfere with a previous release, we use the GitFlow strategy (Driessen 2010) and maintain two long-lived branches named "master" and "dev". The master branch contains only code that is released, whereas all work-in-progress is maintained in the dev branch or in a sub-branch of the dev branch. Once a milestone's tasks are complete, the dev branch is merged into master, and that commit is tagged as a "release".

Pull Requests: Pull requests (PRs) are GitHub's mechanism to let a contributor inform other contributors that they wish to merge changes in their own branch into another branch. We require PRs to merge changes to the master or the dev branch on the server (DASSL 2018d). All PRs require approval from at least two other contributors, but it is common for all team members to review every commit. Every PR is expected to leave the dev branch stable with exceptions clearly noted in PR comments and discussion.

Code reviews: We review all code and non-code artifacts, including test scripts, prior to merging into the dev branch. Examples of issues identified in reviews include repeated and overly-verbose code, typographical errors, formatting inconsistencies, and unclear internal documentation. This attention to detail often results in many revisions and review cycles, but we have found these cycles to be very productive and illuminating for both the submitter and the reviewers. Also, frequent commits, each with meaningful and incremental changes in an "agile" fashion, make reviews easier and more productive.

Testing: We require any code that implements new functionality to include corresponding unit tests, and code that modifies existing functionality to also update existing unit tests. We also require that all code revisions pass both existing and new unit tests after the dev branch is locally merged into the branch of interest. Both the code submitter and all reviewers perform these tests in their own local repositories as part of code review. Presently, unit tests are run manually, and we are investigating test automation.

In addition to unit tests, we also have a separate test suite which performs a full system test on the privilege levels that each kind of ClassDB user has.

Agile-like development: The combination of short development cycles, milestone-driven development, and issue-driven development has the effect of causing incremental changes to the product. Table 2 shows a summary of activity for each version of ClassDB. The data for versions 2.0 and 2.1 shows the number of PRs match almost one to one with the number of issues addressed (defects fixed plus enhancements made). The table also reflects the conscious change we made from Version 1.0 to 2.0 to reduce the number of issues addressed in each PR.

	V1.0	V2.0	V2.1	V2.2
Commits	355	204	234	60
Branches	51	30	21	14
Pull requests	52	29	23	14
Defects addressed	62	31	11	5
Enhancements	NA	2	12	7

	V1.0	V2.0	V2.1	V2.2
Tables	3	4	3	3
Attributes	34	34	15	15
Functions	25	59	84	85
Views	0	14	16	16
Triggers	2	6	7	9

Table 2: Activity across ClassDB versions **Table 3: Number of ClassDB objects by type**

Table 3 illustrates an effect of our agile-like approach on the functionality added with each release of ClassDB, using the number of database objects as a proxy. The significant increase in the number of functions and views from Version 1.0 to 2.0 is mostly due to the large number of API shortcuts added to slice user activity logs, all of which are based on just three functions and one view. The significant drop in the number of table attributes from Version 2.0 to 2.1 is due to a persistent table being replaced with a temporary table as part of improving product quality.

Figure 1 shows how LOC is distributed across production and test scripts, clustered by product version. The significant increase in LOC from Version 1.0 to 2.0 corresponds to the growth in the number of functions and views. Section 5, specifically the discussion related to Table 4, further analyzes the data in Figure 1.

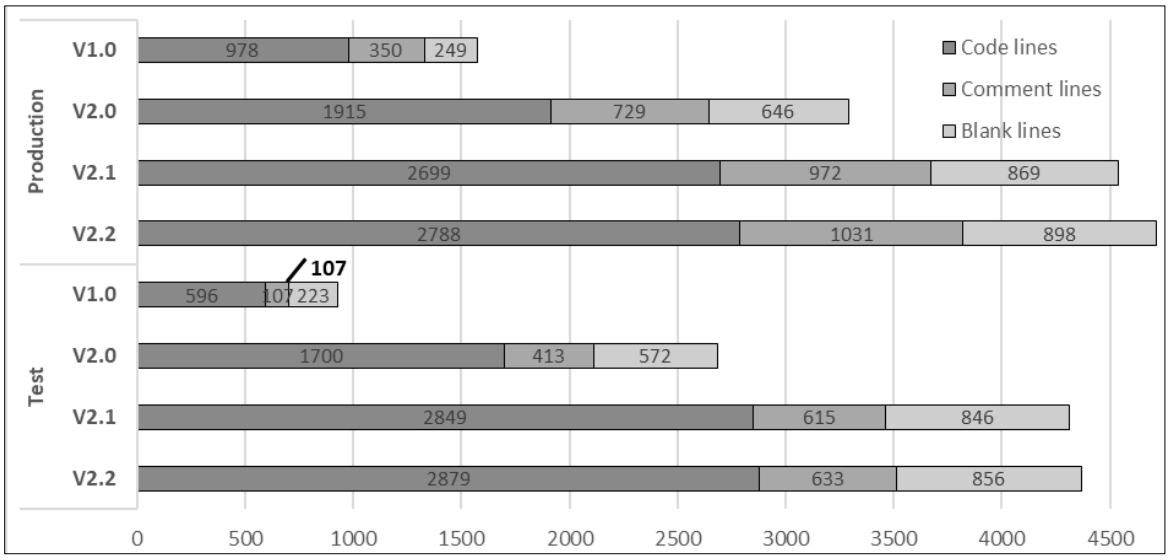

Figure 1: Distribution of LOC in ClassDB by file purpose (production and test), clustered by version

4 Team Culture

We now share the ethos consciously practiced and improved at DASSL. These are the principles that guide DASSL and enable its progress.

Mindset is a skill; the team is a product: As outlined in Section 2.2, students join DASSL any time between their sophomore year and senior year, leading to a large variation in skill levels among them. Thus, DASSL aims to help students develop a *mindset* they can use to produce high-quality work regardless of their skill level. This is done by encouraging a strong team culture where everyone has equal standing and anyone can take leadership of any task. Also, all members are free to (and are encouraged to) participate in the planning, development, and review process of any part of a DASSL product.

Acknowledge others' contributions and provide feedback: Team members are encouraged to interact with other members in a collaborative manner, providing honest suggestions, concerns, and complements as appropriate. This discourse is aided by GitHub Issue and PR systems which allow members to give feedback on others' work. ClassDB PR #214 (DASSL 2018d) is an example of this approach in practice.

Document everything: Everything is documented: code via comments, function usage and API in docs, data and software design, development methods, design rationale, and so on. This effort is aided by the ClassDB Wiki (DASSL 2018e) which allows any contributor to easily create and update docs.

The “documenting everything” approach greatly improves productivity. Because there are large time gaps between development cycles, the extensive documentation allows contributors to quickly refresh their memory, both in terms of how ClassDB works (external docs) and what the code does (internal docs).

Think first, code later: DASSL strongly encourages discussing and planning changes prior to coding. We have several examples of practicing this philosophy, including large-scale changes that have been discussed for days or weeks before implementation.

An example is the addition of disconnection activity logging to ClassDB in response to Issue #206 (DASSL 2018b). At its core, this enhancement required a single line of code that changed a Postgres configuration parameter, but several other changes were needed to fully implement disconnection logging. After an extensive analysis of the issue, we concluded that the best approach was to add several new columns to a particular table. In the process, we identified several additional issues such as `NULL` constraints on one of the new columns which required special attention to porting data in existing ClassDB installations. By identifying such issues prior to implementation, we were able to come up with a robust design that helped us easily add new features and also upgrade existing ClassDB installations.

Focus on building systems, not on writing programs: Although students develop core CS and programming skills at DASSL, end-to-end engineering is the DASSL objective. An example of this approach is our effort to maintain a property called **idempotence** in all ClassDB scripts which allows a script to be safely run any number of times. Running an idempotent script for the first time makes all necessary additions and modifications but re-running does not make additional modifications or cause errors.

In addition to greatly reducing testing effort, script idempotence is beneficial to users because it permits easy installations and updates without having to remove any existing installation, and it is generally more forgiving than non-idempotent code. Further, maintaining idempotence improves code quality because it increases our awareness of what each line of code does.

Code like the world is watching (because it is): We expect to display not only our final product, but the discussions and processes we use to get there as well. Posting code in a public repository and using various GitHub features to discuss and comment our decisions compels us to be conscious of our actions and decisions. For example, we closed ClassDB Issue #233 due to its changes being outside the scope of the current milestone, but we left the discussion public so all could see the reason behind our decision.

Always be learning, adopting, and adapting: Every member of DASSL is encouraged to continuously learn new tools, methods, and then to use their learning to address new issues as well as to improve existing solutions. This approach has helped us adopt new tools and methods throughout the development of ClassDB. For example, during the development of ClassDB 2.0, we quickly evaluated GitFlow and switched to it from a standard Git workflow. Similarly, when the university provided us Microsoft Teams (Microsoft 2018), we quickly evaluated it and incorporated it into our process starting with ClassDB 2.1.

Reciprocal, equitable, and voluntary participation: These three participation traits play a large part in successful teamwork at DASSL. All three traits are evident in practically every ClassDB issue and PR, the discussion on ClassDB Issue #206 being particularly demonstrative:

- Reciprocal: Team members respond to initiatives from others and return favors when helped by others.
- Equitable: On the whole, every team member adds equal value. Members adding too little may be viewed as not contributing their fair share; those adding too much could be perceived as dominating.
- Voluntary: Every team member willingly participates in team activities and does not feel they are being forced.

Reciprocal and equitable participation occasionally adds a bit more work to all team members, but in our experience, it also tends to produce better solutions faster, keep every team member more aware of the product details, and generally strengthens the sense of team and teamwork.

5 Focus on Quality

In this section, we discuss two specific aspects of our effort to produce high-quality software.

5.1 Maintainability

At a high level, DASSL emphasizes the following code characteristics and enforces them during reviews:

- Function and block structure: Organize code into functions and blocks; avoid lengthy functions.
- Code format: Choose identifier names to reflect their purpose; consistently indent code.
- Code chunks: Write code in chunks; perform one related task in each code chunk; keep chunks small.
- Comments: Comment each code chunk; say something that is not obvious in the code.
- External documentation: In the documentation, describe every API entry point available to end users.

We now present a brief quantitative analysis of LOC data to illustrate the impact of enforcing the code characteristics outlined. We use data generated from the FOSS counting tool *cloc* (Danial 2018) in this analysis.

Table 4 presents a summary of the number of files and LOC in production scripts broken down by version and line type. The Count section of Table 4 shows the raw number of files and lines, with values in parentheses showing percentage growth over the previous version. For example, the number of code lines grew 96% Version 1.0 to 2.1, but it grew only 40% from Version 2.0 to 2.1.

The Distribution section shows the proportion of each line type: Generalizing across versions, 59% of all lines are code lines (not comment or blank); 21% of lines are comments; and 20% of lines are blank. The Ratio section provides an alternative view of this distribution showing that across all versions, the source files contain one comment line and one blank line for approximately every three code lines.

Table 4 quantitatively supports our approach to writing code in small chunks and commenting chunks. The consistency of the ratios across four versions also shows the consistency of our coding and review practices. It is important we note that we did not set out to consciously arrive at the specific ratios seen in Table 4. In fact, this data was gathered for the first time only after ClassDB 2.1 was released.

The Density section of Table 4 shows the average number of lines of each type per production script, with the minimum and maximum number of lines shown in parentheses. Although the average values indicate small files, which in turn point to higher maintainability, the value ranges indicate the existence of “heavy hitters”. Indeed, an analysis of Version 2.1 shows that just four scripts contribute 66% of all code lines and provides us a list of candidate files to refactor. However, refactoring files can be non-trivial because that also requires refactoring unit tests, and potentially documentation as well in some cases. It will also likely change the dependencies between scripts and thus require a revision to the product installation process.

	Version 1.0	Version 2.0	Version 2.1	Version 2.2
Count (growth % from previous version)				
Files	12	23 (90%)	24 (4%)	26 (8%)
Code lines	978	1915 (96%)	2699 (40%)	2788 (3%)
Comment lines	350	729 (108%)	972 (33%)	1031 (6%)
Blank lines	249	646 (159%)	869 (34%)	898 (3%)
Total lines	1577	3290 (108%)	4540 (38%)	4717 (4%)
Distribution: % of total lines				
Code lines	62%	58%	59%	59%
Comment lines	22%	22%	21%	21%
Blank lines	16%	20%	20%	20%

Ratio of non-code lines to code lines				
Comment to code	1 per 2.8	1 per 2.6	1 per 2.7	1 per 2.7
Blank to code	1 per 3.9	1 per 3.0	1 per 3.1	1 per 3.1
Density: average LOC per file (also min-max LOC)				
Code lines	82 (4-284)	83 (2-347)	112 (2-610)	107 (2-610)
Comment lines	29 (7-64)	32 (8-134)	41 (8-163)	40 (8-152)
Blank lines	21 (3-69)	28 (4-109)	36 (4-184)	35 (3-184)
Total lines	131 (14-417)	143 (16-590)	189 (14-957)	181 (13-936)

Table 4: A summary of LOC data for ClassDB production code (corresponds to the first cluster in Figure 1)

5.2 Issue Management

We now present how we organize issue reports to help us analyze product quality and prioritize work.

We expect team members to log every potential issue they observe and require each issue report to include the following parts (the first three parts are required):

- Title: For defects, the title concisely describes the problem; for enhancements, it states the requirement.
- Description: The description clearly states the problem and where possible include links to the issue location. We also encourage submitters to include proposed solutions in issue descriptions.
- Nature: Each defect is classified as Wrong, Missing, or Extra (or a combination). Enhancements and non-issues are explicitly tagged as such. We use issue nature to summarize how many defects are addressed and how many enhancements are made in a release. (See Table 2.)
- Related issues and PRs: Citations that tie the issue to other issues or PRs so the issue can be addressed comprehensively.

As part of issue management, the lab director assigns the following additional information to each issue. We do not prevent any member from changing any of the assigned metadata, but prefer (and encourage) all members debate any assignment in issue threads before changing the metadata:

- Priority: The urgency (high, medium, low) with which the issue needs to be addressed.
- Milestone: The product milestone in which the issue is to be addressed.
- Pipeline: A coarse-grained issue bucket, a feature available in ZenHub.

We presently do not assign severity and effort estimate to issues, but we are incrementally introducing these and other concepts to the team.

As summarized in Table 2, we have thus far addressed 118 issues in ClassDB, including 104 defects and 14 enhancements. However, we emphasize that the count of 104 defects does not indicate poor quality, because only 42 of those defects are logged after the first release, and none of those defects have been “show stoppers”. In fact, to date, we have seen only one issue (Issue #200) in the deployed system that prevented the use of a part of a system. We consider this a “low severity” issue because it did not impact the overall system functionality or usability, and its occurrence required a power outage, disk crash, or other rare event. Yet, we addressed this issue as high priority because it prevented the use of a particular function an instructor is likely to perform frequently.

6 Perspectives

We now present individual perspectives of the three authors about their experience at DASSL. The writing in the rest of this section is intentionally in first-person singular.

6.1 Andrew Figueroa (student)

DASSL presents its members with unique opportunities to analyze, build, and maintain data-intensive systems in ways that are not possible in a classroom environment, or even an industry internship. Much of what leads to this has been described earlier in this paper. However, one aspect not yet mentioned is that by participating in DASSL, students such as myself are given real responsibility of a major project.

Classroom experiences are an inherently "sheltered" environment: instructors generally have a solution in mind and can provide detailed feedback when students are headed in the wrong direction. Tasks have a specific set of requirements that must be met. Additionally, there is only one element at risk if an attempt goes wrong: a student's grade. Although grades are an important aspect, the effects of a single failed attempt are often remedied by the ability to make multiple attempts and by the weight of other assignments.

However, experiences at DASSL differ in that there is no immediately clear potential loss: all work performed is voluntary, ungraded, and not as time-sensitive as in a classroom. However, this is replaced with the responsibility of collaboratively undertaking research or developing a system that will undergo an actual publication or release. A mistake could result in the whole group having to later lose precious time fixing it. Even worse, some mistakes could result in major research efforts being for nothing, or security vulnerabilities in production software. Additionally, at least for our software projects, the entire process is publicly viewable. At times this fact is stressful, but this is also a significant part of what makes the whole experience as educational and enjoyable as it is.

Responsibility, among the other aspects of DASSL's culture, encourages me to always gradually improve as opposed to simply meeting the requirements (which for the types of projects implemented in DASSL, are usually either non-concrete or are set by us in the first place). The effects of this encouragement can be seen through the consistent quality improvement of code, documentation, and discussions in ClassDB contributions in which I have been involved.

One of my early contributions to ClassDB was Commit `494ed85` as part of PR #9. Although I had a basic understanding of using Git and GitHub, this commit shows several practices I now avoid, such as a non-specific subject line and including multiple unrelated changes in a single commit. Also, this PR itself has several issues such as having over 50 commits with major changes to a script and changes to unmentioned files. Contrast this PR with the later PR #92 in which all included commits were related to each another, had descriptive subject lines, and each commit made just one significant change. PR #92 also includes a detailed description of the changes made, their effects, and potential situations where the changes may behave unexpectedly.

Beyond developing software products, DASSL has encouraged me to participate in writing several papers that summarize the work done at DASSL, including this paper. I would otherwise have not even considered writing any academic papers during my undergraduate years. Beyond helping to further develop my understanding of the topics at hand, these papers have presented me with several opportunities that I would otherwise not have even imagined having, not the least of which being attending and presenting at an ACM conference in London, UK (Murthy, Figueroa, and Rollo 2018). Prior experience presenting new topics to other DASSL members and presenting DASSL work to other students and faculty at WCSU helped prepare me for this international conference and led to it being a success.

6.2 Steven Rollo (student)

I strongly feel DASSL helped me develop a competitive software-engineering skillset by providing opportunities to participate in challenging real-world projects. Learning the modern software-engineering practices described in this paper is attractive to any employer because modern workflows are based around these tools and techniques. While it is possible to grasp basic tool usage in the classroom, developing mastery requires practice in real-world scenarios.

DASSL provides a unique opportunity for students to extensively learn and practice a full range of software-engineering techniques with strong emphasis on teamwork and team communication, which I found to be particularly valuable. Prior to my participation in DASSL, I put little effort into developing an effective teamwork skillset. Between never being assigned group work in class and only working on relatively small personal projects, it felt unnecessary to develop these skills. The Summer 2017 session of DASSL placed a large emphasis on effectively documenting work, communicating issues, and developing a working relationship with other team members. Learning these skills allowed me to function effectively in the team projects I participated in at DASSL.

My experience at DASSL also shows that learning these skills requires practice and is evident in the progression of my work at DASSL. For example, large improvements in clarity can be seen between Issue #18 and Issue #243 I logged respectively during Milestones 1 (2017) and 3 (2018) of ClassDB. The title of Issue #18 ("The object name should be recorded in the student table on sql_drop events") is relatively unhelpful. Also, the problem stated in the issue description is only tangentially related to the title. Overall, though this issue does contain helpful information, it failed to communicate the actual problem effectively. In comparison, Issue #243 is much more effective. This issue's title explicitly states what exactly is wrong with the product. The issue description then provides a diagnosis of the problem, an example of what causes the problem, and three possible solutions. I was able to gain the ability to communicate in such a mature way only due to the practice from my experience at DASSL.

I feel these improvements in communication skills greatly helped me when I looked for a job after graduating from WCSU. My ability to communicate effectively has made me more confident about my ability to work on a team, instead of just being a lone programmer. When interviewing for a job, I was able to easily convince others that I have experience working in an agile environment. Additionally, I was able to quickly grasp the workflow of the department I was interviewed at, which allowed me to make a more informed choice about my suitability to work there.

6.3 Sean Murthy (faculty member)

I believe the student perspectives in Sections 6.1 and 6.2, and similar feedback from other students, testify to DASSL's effectiveness in preparing students for software-engineering careers by helping them learn and practice modern tools and processes of software engineering. I am pleased to lead this effort.

I feel my experience in commercial software development as well as my experience teaching a variety of CS topics have been helpful in leading DASSL. The following factors have also played a significant role:

- Spend much of summer and winter breaks with students: I spend about 80% of the break with students.
- Work long hours, about 8 hours a day: For example, I spent about 115 hours on just development and management of ClassDB 2.0 (over 5 hours a day). In addition to product development and management, I customize plans for each student because every student learns differently and is motivated by different things. I also meet each new student member one-on-one about one hour each week.
- Lead by example: Participate in every aspect of product development and management; submit own work for review; share critical review of own work to set a model for students and to make students feel comfortable enough to submit their work for review.
- Earn and maintain student trust: Always have student interest in mind and practice it visibly in both words and deed. No action or activity should hurt students' main purpose of being at a university. No activity can be solely to the benefit of a faculty member unless separate and appropriate employment terms are negotiated with students.
- Continuously learn new tools and techniques.

I feel the DASSL process is repeatable but is not necessarily scalable or easily sustainable because It takes considerable amount of time to provide high-quality mentoring while also producing high-quality work. Also, DASSL tasks are in addition to publication, teaching preparation, administrative tasks, family time, and other commitments that need equal or greater attention.

The following factors are also potential hindrances to scalability and sustainability:

- Small student pool: Many students are interested in the benefits of DASSL, but few are able to spend the necessary effort, largely because they need to work or they do not live on or near campus.
- Different engagement mode for new members: Online meetings can help enlarge the student pool, but intense daily in-person meetings are necessary for new students. Additional online engagement with seasoned students requires additional time and energy on the part of faculty members.
- Large portfolio: The lab needs to start a new product or module about every two years in order to expose students to different stages of product life cycle, which results in a large product portfolio over time.
- Short engagement and availability: Engagement is only over summer and winter breaks, and students spend at most four term breaks in DASSL, which results in frequent changes to the team composition.

The following actions can address the aforementioned challenges to scalability and sustainability:

- Increase faculty participation: Small faculty size makes additional faculty participation hard. However, different faculty members can (and do) organize DASSL-like effort. For example, Dr. William Joel involves students in his Graphics & Interactive Technologies Research Group at WCSU (Joel 2010).
- Maintain “anchor students”: Anchor students are those with commitment to quality and have the industry to learn a variety of topics. It is especially helpful if anchor students join the effort soon after their third semester so they are around to gain extensive practice which they can use to lead sub-projects and semi-independently assist new student members.
- Offer paid positions: Paid positions recognize students who contribute significantly to the cause. They also permit students to continue in the lab instead of pursuing “non-tech jobs” elsewhere for sustenance.
- Raise funds: Stipends can increase both faculty and student participation, but stipends obviously require funding. Funding can realistically be from prospective employers who stand to gain from these efforts.
- Add an “engineering bent” to the academic program: Most CS courses rightfully focus on programming, algorithms, architecture, and other core concepts. Yet, there is scope for introducing the concepts of requirements, design, milestones, and other engineering aids early in the academic program. I use this approach for term projects in the CS205 and CS305 courses I teach (WCSU 2013).
- Offer an early “DevOps” course: A course targeting 4th-semester students to introduce topics such as version control, testing, issue management, and continuous integration can be quite helpful. Such an early course provides students additional opportunities to practice “DevOps” in labs and in later courses in the academic program. I am scheduled to teach such a DevOps course in Spring 2019.

In short, the continued success of DASSL and similar efforts requires support from university administrators and the industry. The combined effort can improve the quality of CS education and of software in the large.

7 Summary

Employers expect employees to possess an "Agile and DevOps mindset" and produce “high quality” software using sophisticated tool chains. Increasingly, they do not exclude even new CS graduates from this expectation. However, undergraduate CS programs are structurally unable to produce new graduates who meet this requirement. Hands-on co-curricular labs can fill this gap by introducing students to key aspects of building high-quality software and help meet employer expectations. Alternatively, employers will likely prefer new graduates who possess these skills over those who do not.

Being such a hands-on lab, since Spring 2017, DASSL has infused 12 undergraduates with many of the modern skills and the mindset employers seek. It does so by engaging students in every stage of the product life cycle using much of the same processes and tools professionals use. In the process, students collaborate with faculty members to build and maintain non-trivial products that meet a relatively high standard of quality. As illustrated in Sections 2-5, ClassDB is one such high-quality product DASSL has produced.

Efforts such as DASSL are reproducible and repeatable, but are not easily scaled or sustained. Their continued success depends on a sustained “critical mass” of compatible students, and on faculty members with significant experience building high-quality production software. Also, both faculty and students need to spend (or lose) considerable amount of time, energy, and money outside regular academic commitment. Small changes in CS programs and support from administrators and industry beneficiaries can address these and other impediments.

8 Acknowledgments

We thank the faculty and administrators of Western Connecticut State University, particularly the CS department, for encouraging DASSL. We thank Kevin Kelly, a student member of DASSL, for assistance with compiling some of the data included in this paper. We are also thankful for the generous educational licensing policies of GitHub and ZenHub. Finally, we thank the reviewers of this paper for their feedback on drafts.

References

Creative Commons. 2013. “CC BY-NC-SA 4.0.” Creative Commons - Attribution-NonCommercial-ShareAlike 4.0 International. 2013. https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.

Danial, Al. 2018. *Cloc*. https://github.com/AlDanial/cloc.

DASSL. 2017a. “DASSL GitHub Repositories.” GitHub Organization. 2017. https://github.com/DASSL.

DASSL. 2017b. “Data Science & Systems Lab Home Page.” 2017. http://dassl.github.io.

DASSL. 2018a. “ClassDB Documentation.” 2018. https://dassl.github.io/ClassDB/.

DASSL. 2018b. “ClassDB Issues.” 2018. https://github.com/DASSL/ClassDB/issues.

DASSL. 2018c. “ClassDB Milestone M2.” 2018. https://github.com/DASSL/ClassDB/wiki/Milestone-M2.

DASSL. 2018d. “ClassDB Pull Requests.” 2018. https://github.com/DASSL/ClassDB/pulls.

DASSL. 2018e. “ClassDB Wiki.” 2018. https://github.com/DASSL/ClassDB/wiki.

Driessen, Vincent. 2010. “A Successful Git Branching Model.” January 5, 2010. https://nvie.com/posts/a-successful-git-branching-model/.

Ferraiolo, David F., and D. Richard Kuhn. 1992. “Role-Based Access Controls.” *National Computer Security Conference* 15: 554–63.

Joel, William. 2010. “Graphics & Interactive Techniques Research Group.” January 5, 2010. http://www.wcsu.edu/cs/sample-page/computer-science-research/computer-science-grg/.

Jones, Capers. 2007. *Estimating Software Costs: Bringing Realism to Estimating*. 2nd Edition. McGraw-Hill Education.

Microsoft. 2018. *Microsoft Teams*. Microsoft Corporation. https://products.office.com/en-US/microsoft-teams/group-chat-software.

Murthy, Sean. 2018a. “Employing FOSS Tools to Improve Learning and Increase Opportunities.” Presented at the Northeast Regional OER Summit 2018, University of Massachusetts, Amherst, MA. https://drive.google.com/drive/folders/1I8gd_dQEIzgfaawltCXgXL6vXazgvlb3.

Murthy, Sean. 2018b. “Shared Governance by Design: Employing FOSS Tools to Improve Learning, Reduce Cost, and Increase Opportunities.” Presented at the 4th Annual Conference on Shared Governance and Student Success, Southern Connecticut State University, New Haven, CT, April 13.

Murthy, Sean, Andrew Figueroa, and Steven Rollo. 2018. “Toward a Large-Scale Open Learning System for Data Management.” Presented at the Fifth Annual ACM Conference on Learning @ Scale. London, UK. https://doi.org/10.1145/3231644.3231673.

PostgreSQL Development Group. 2017. “PostgreSQL 9.6.9 Documentation.” 2017. https://www.postgresql.org/docs/9.6/static/index.html.

Wallshein, Corinne C. 2010. “Estimating Software Effort Hours for Major Defense Acquisition Programs.” Dissertation, George Mason University. http://ebot.gmu.edu/bitstream/handle/1920/6025/Wallshein_Dissertation_Spring_2010.pdf.

WCSU. 2013. “Computer Science Course Catalog.” 2013. http://www.wcsu.edu/catalogs/undergraduate/sas/courses/computer-science/.

Whitworth University. 2018. “Mathematics & Computer Science Course Catalog.” 2018. http://catalog.whitworth.edu/undergraduate/mathcomputerscience/#courseinventory.

ZenHub. 2018a. “An Intro to ZenHub Epics.” 2018. https://help.zenhub.com/support/solutions/articles/43000010341-an-intro-to-zenhub-epics.

ZenHub. 2018b. “ZenHub.” 2018. https://www.zenhub.com/.