A Software Engineering Capstone Infrastructure that Encourages Spreading Work Over Time and Team

***Abstract*—How can instructors facilitate spreading out the work in a software engineering or computer science capstone course across time and between team members? Currently teams often compromise the quality of their learning experience by frantically working before each deliverable. Some team members further compromise their own learning, and that of their colleagues, by not contributing their fair share to the team effort. To mitigate these problems, we propose using a GitHub template that contains all the initial infrastructure a team needs, including the folder structure, text-based template documents and template issues. In addition, we propose each team begins the year by identifying specific quantifiable individual productivity metrics for monitoring, such as the count of meetings attended, issues closed and number of commits. Initial data suggests that these steps may have an impact. In 2022/23 we observed 24% of commits happening on the due dates. After partially introducing the above ideas in 2023/24, this number improved to 18%. To measure the fairness we introduce a fairness measure based on the disparity between number of commits between all pairs of teammates. Going forward we propose an experiment where commit data and interview data is compared between teams that use the proposed interventions and those that do not.**

***Index Terms*—software engineering; capstone; template repository; productivity measures; fairness metric**

I. Introduction

The workload for a software engineering or computer science capstone team project is often unevenly distributed over time and between team members. Teams typically work in frantic bursts of activity right before a deadline and then cease almost all activity until their next deadline. These work habits compromise the learning objectives of the course because the students do not have time to properly plan their activities or reflect on their work. The uneven distribution of effort between team mates is also problematic. Some students take on an unfair share of the work, causing them stress and possibly hurting their experience in other courses, while those investing less effort (so-called free riders [1]) miss important learning opportunities. How can instructors mitigate these problems?

To address the uneven distribution of work, we need to first think about why the problems exist. A student project is not the same environment as the workplace. Students are often learning the content just before applying their knowledge. Since they are doing a capstone project for the first time, students might struggle with determining the expectations; they might not know where to start. In industry team members usually dedicate most of their time to a project, unlike an academic environment where students are juggling many courses [2]. Moreover, team members cannot be fired or moved to another project for poor performance. Peer pressure and the prospect of uncomfortable social interactions can make it challenging for someone to take charge of their group, or to criticize other group members.

For saving the student's time and clearly showing expectations, we suggest requiring all capstone teams to start from the same GitHub template repository. The template repository is populated with folders, text-based template documents and template issues. [ADD Citations on this topic]. Too much freedom in decision making can be paralyzing for a new team. They don't know where to begin. By making the infrastructure decisions for them, they can focus on their project.

An even workload between teammates can be improved by early awareness of potential problems. Ideally, the teams should plan how to deal with problems, before the problems occur. We propose doing this by identifying quantifiable productivity metrics for each team member (like counting meetings attended, issues closed, and commits) and having the team write a team charter at the beginning of the term that lays out their unambiguous expectations. The idea of a team charter is not new [3], [4], [5], but as far as we are aware, we are the first to suggest incorporating specific quantifiable GitHub-derived metrics and consequences. Other studies have used commits to understand/explain team behaviour a posteriori [6], [1], but having the students actively collect and use this data during the course appears to be a new idea.

In Section II we describe the baseline structure of the capstone course. We propose two interventions to the baseline: 1) using a template repository; and, 2) explicit quantifiable team contribution measurement. We follow this with some encouraging preliminary data from when we partially introduced the two interventions into the course (Section III). We then describe our proposed approach for collecting more detailed data to judge the effectiveness of the proposed interventions (Section IV). The presentation of the proposed experiment includes discussion of threats to validity.

II. Baseline and Proposed Infrastructure

The infrastructure described here matches the final year SE capstone course at [Redacted]. spanning a full year, being a group project, having an implementation as its end deliverable, having a customer for each project, and including student reflection [7]. This course is currently delivered to 150 students divided into 29 groups of 4–5 members (the typical size for

capstone courses [8]). Teams are provided with a list of curated software development projects from academia and industry. Teams can also propose their own projects. Most projects have a supervisor/client that the team can meet with to discuss their project. In cases where there is no supervisor, explicitly identifies the relevant stakeholders/users for their project.

A. Structure and Timeline

Figure 1 show the V-model [9] structure of the capstone course. The documents created include a Software Requirements Specification (SRS) and Verification and Validation (VnV) plans and reports. Due to time constraints, not all artifacts of the V-model are produced. Those that are created are circled with red ellipses, along with an annotation showing the week number where the artifact is due for a full year (26 week) course. The week is when the Revision 0 draft of the document is due. Almost all documents also have a second revision (Rev1 Doc) that is due at the end of the course (Week 26). The iteration allows students to take the formative assessment for Rev 0 to produce a higher quality document for their summative review. An iterative process for software capstone course is recommended by VanHanen and Lehtinen [10] to improve the learning outcomes. Although there are reasonably frequent interactions with the TA and instructor, we do not follow the agile process recommended by some [11], [12].

In recognition of the value for teams of "getting their hands dirty", a Proof of Concept (POC) Demo is scheduled for week 10. During this demo the teams demonstrate the aspect of their project that is of most concern for feasibility of the project, providing an opportunity to revise the project scope if necessary. The Rev0 demo is expected to show off the final and complete product. The teams rarely achieve this, but the push for Rev0, together with the feedback they received, allows them to improve their software for the final demo (Rev 1 demo). The structure of the course is stable, having been offered in this form for four years. The interventions described in the next sections are in the context of this structure.

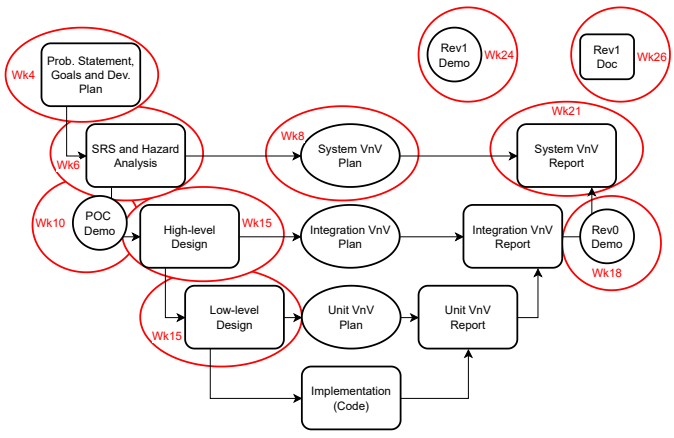

Fig. 1: V Model Used for Capstone Deliverables

B. Template Repository

All teams start their project by using the same GitHub template repository. The template repo, summarized in Figure 2, contains all the initial infrastructure each team needs, including the folder structure, text-based template documents and template issues. The goals of the template are to remove the time team's spend building their project's infrastructure, and to standardize all the arbitrary decisions, like folder and document names, between all the teams. The standardization helps teams when doing peer reviews of each other's work and it improves communication between the teams, teaching assistants and instructor. When students have a clear idea of the expectations, they should find it easier to dive into their project.

The template documents are written in LaTeX, although teams are allowed to redo the template in another text-based format, like Markdown, if they wish. Besides the advantage of separating document appearance from document content, the text-based format facilitates tracking the productivity of the team members through git commits, as discussed in Section II-C. The documents correspond to the deliverables in Figure 1. The students can use any standard SRS template, including selecting one of the three options given: SRS (a template for scientific computing software [13]), SRS-Meyer (a template by Bertrand Meyer [14]) and SRS-Volere (the Volere template [15]).

For further standardization, the template repo includes four issue templates for: 1) team meeting agendas; 2) TA-team meeting agendas; 3) supervisor-team meeting agendas; and, 4) lecture attendance. In addition to encouraging good organizational habits, the issues are also used to partly measure the commitment of students to their teams, as discussed in the next section.

C. Team Contribution Measurement

To improve the distribution of the workload to all team members, we can take advantage of the quantifiable productivity measures available from git and GitHub. The value of a metrics-based software engineering process is emphasized by Conn [2], although they do not list their recommended metrics. In the current work the suggested metrics for each team member are counting team meeting attendance and using GitHub insights to count the commits to the main branch. More complex metrics are available, like lines of code, function points, use case points, object points, and feature points [16], but by default we keep things simple and standard between teams. However, if a team desires more complex metrics, they can measure those alongside the required ones. Teams are reminded that if they work together on something, they can use co-author commits. Each team produces a summary table as part of their performance reports, which are produced before the three demonstrations: POC demo, Rev0 demo and Rev1 demo (see Figure 1 for the timing of the demos). In the performance report the team can record an explanation for why a team member appears to perform poorly on any of the

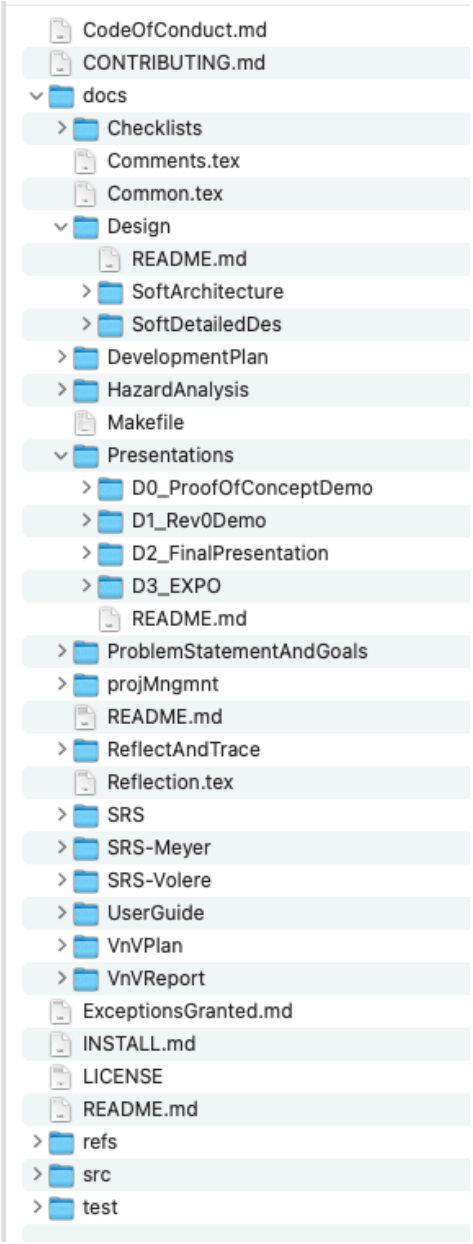

Fig. 2: GitHub Capstone Template

metrics. For instance, a team member may have focused their commits on a branch that has not yet been merged into main.

The teams set specific expectations for their team members in their team charter. For instance, the team might have a rule that missing 20% of the team meetings before the proof of concept demonstration requires the offender to pick up the coffee for the next team meeting. A more serious rule might be something like, if a team member has less than 5% of the total team commits before the POC demo, the team will schedule a meeting with the course instructor to discuss the problem. The encouraging feature a priori creation of rules is that the difficult discussion happens while relationships between team members are likely strong. If a problem later occurs a team member doesn't have to muster the courage to say that they are concerned with a colleague's performance, instead they can point to the team charter and highlight the relevant, already agreed upon rule.

The hope is that explicitly capturing productivity measures during the term will reveal any problems with team collaboration. Ideally the problems will be revealed early and improved, but if the problem cannot be dealt with, at least there will be enough data to assign a fair individual grade to all team members. Although by default all team members share the same grade on a deliverable, this can be multiplied by a "team contribution factor" if the data suggests this is necessary for fairness. The data is not just commits. Any change in grades needs to be supported by feedback from the TAs, feedback from supervisors, instructor observations, and anonymous team surveys. The team contribution factor penalty is generally only applied after a team member has had an explicit warning from the instructor.

TABLE I: Time-Spread Metrics Across Two Classes

Metric	**2022/23 Value**	**2023/24 Value**
Total Commits	6140	5120
Total Days	243	244
T-0 Days	10 (4.12%)	10 (4.10%)
T-0 Commits	1471 (23.96%)	942 (18.40%)
T-2...T-0 Days	30 (1.37%)	30 (1.37%)
T-2...T-0 Commits	2377 (38.71%)	1872 (36.56%)

III. Preliminary Data

Look at commits over time, and possibly lines (removing outliers) of code over time

A. Timeline Comparison

To measure the spread of work across time, we propose the following metrics:

1) Daily commit graphs (examples for 2022/23 and 2023/24 are shown in Figs. 3 & 4).
2) T-0 Proportion: The proportion of commits made on major deliverable due dates
3) T-2...T-0 Proportion: The proportion of commits made on major deliverable due dates and in the two days prior to them.

A summary of results in 2022-23 and 2023-24 is shown in Table I. We observed similar results between the two years. There was a slight reduction in the T-0 and T-2...T-0 commits between these two years, especially the T-0 commits. It is clear that there is still work to be done to encourage spreading out work, but it is likely to be impossible to ever fully eliminate this effect because of the nature of due dates in students' busy schedules, but improvements in these metrics would show progress towards the goal of spreading out work across time more effectively.

B. Measuring Fairness

To measure the spread of work across team, the fairness of work distribution among teammates, we sought to compute

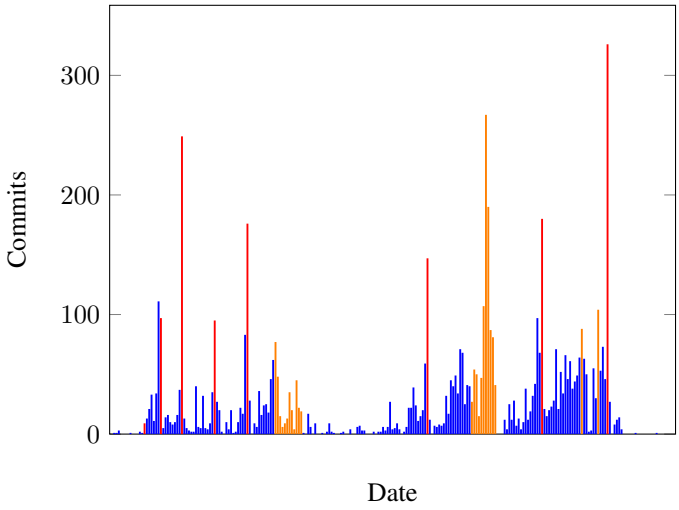

Fig. 3: Timeline of Commits for 2022–2023. Dates shown in red are due dates for major written deliverables, and dates in orange are days where presentations were scheduled.

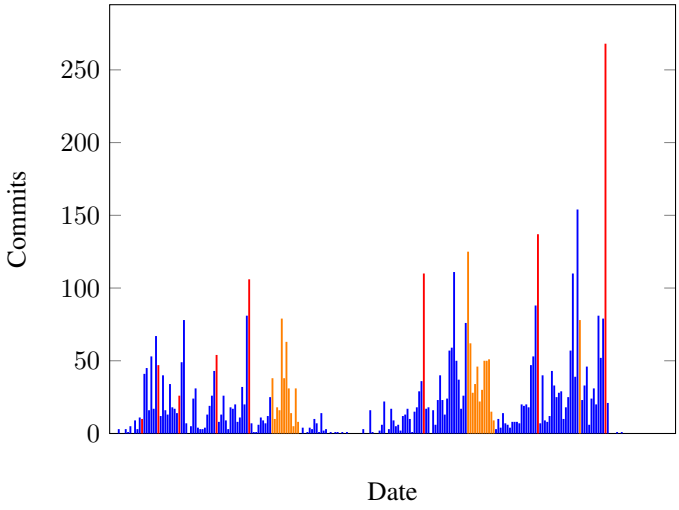

Fig. 4: Timeline of Commits for 2023–2024. Dates shown in red are due dates for major written deliverables, and dates in orange are days where presentations were scheduled.

a metric where values range from 0 to 1, and where values in between contain meaningful information. Thus, we devised the following *unfairness* metric where C:

$$\text{unfairness}(C) = \frac{\sum_{c,x \in C, c>x} (c - x)}{(|C| - 1) \cdot \sum_{c \in C} c}$$

where C is the multiset of teammates' numbers of commits to the repository.

The metric computes the sum of the difference between each teammate's commits and those who committed less than them, normalized by the number of teammates (excluding themselves) and the total number of commits. This yields a value from 0 to 1, called the *unfairness* metric, where:

- 0 indicates that teammates did an equal amount of work
- 1 indicates that all the work was done by one teammate
- A value between 0 and 1 indicates the proportion of work per person which could have been given to someone who did less work

Fairness is defined as $\text{fairness}(C) = 1 - \text{unfairness}(C)$.

For example, if a team with Persons A, B, and C did 10, 5 and 5 commits respectively, then $\text{unfairness}(\{10, 5, 5\}) = 0.25$. This is because on average Person A did 5 more commits than their teammates and thus these 5 commits (out of 20) are considered *unfair work*. The fairness value is thus 0.75.

Figs. 5 and 6 show the fairness values for teams in 2022/23 and 2023/24 respectively.

In the future, we will experiment with applying this metric to things other than commits (e.g. lines of code written, issues created/closed, etc.), as well as investigating a correlation between lower fairness values and perceived unfairness according to the teammates themselves. Another research question would be if having live access to this fairness metric encourages teams to share work more evenly or simply encourages them to "game the system" to increase the value.

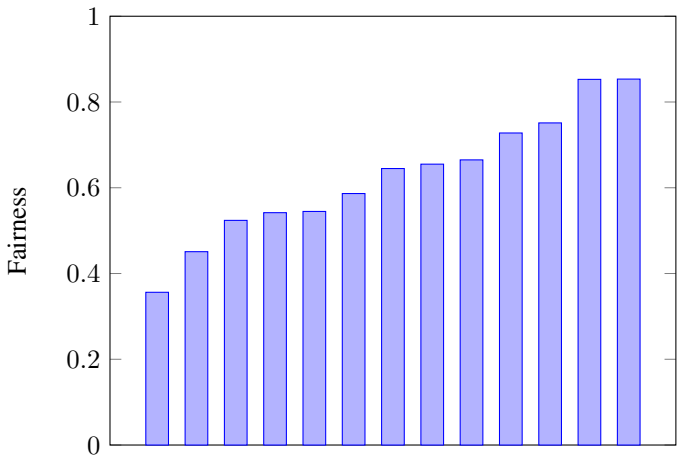

Fig. 5: Fairness of Commits Per Team 2022/23 [n=13] (Mean: 0.63, Stddev: 0.15)

IV. Proposed Experiment

blurb

A. Experiment

Start with research questions.
entry and exit surveys
Collect the same data as in Section III and conduct focus groups in all three CAS capstone courses (SE, CS and TRON).

B. Threats to validity

- Using commits to measure productivity is possibly too simple a measure. Some team members might commit small deltas of work, while others only commit when they

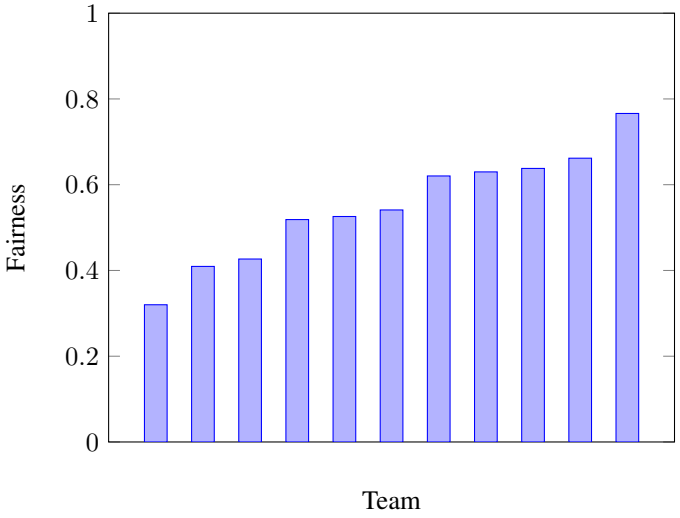

Fig. 6: Fairness of Commits Per Team 2023/24 [n=11] (Mean: 0.55, Stddev: 0.13)

complete a major change. Going forward, productivity may be measured by a count of work events [17], where a work event is a push, merged pull request, issue comment or pull request. review comment

- Multiple changes are made to the course, so it is difficult to determine which change influences the student behaviour. The focus group should hopefully tease that out.
- Comparing different courses with different instructors, different backgrounds for students, etc.
- Not a controlled experiment - introducing more than one change into the course. The changes are related because the productivity metrics would not be possible without a version control system.
- The interventions proposed here might behave differently for a capstone course that follows a different structure (Section II-A).

V. Concluding Remarks

The template presented here is for the capstone course under discussion; the template could be forked and modified to match the needs of a different capstone course.

VI. Data Availability

The raw data used to make these graphs is available on the paper's GitHub repository, along with the scripts used to generate the data.

References

[1] M. Tushev, G. Williams, and A. Mahmoud, "Using GitHub in large software engineering classes. An exploratory case study," *Computer Science Education*, vol. 30, no. 2, pp. 155–186, Apr. 2020.

[2] R. Conn, "A reusable, academic-strength, metrics-based software engineering process for capstone courses and projects," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. Norfolk Virginia USA: ACM, Mar. 2004, pp. 492–496.

[3] J. E. Mathieu and T. L. Rapp, "Laying the foundation for successful team performance trajectories: The roles of team charters and performance strategies," *Journal of Applied Psychology*, vol. 94, no. 1, pp. 90–103, 2009.

[4] W. H. Johnson, D. S. Baker, L. Dong, V. Taras, and C. Wankel, "Do Team Charters Help Team-Based Projects? The Effects of Team Charters on Performance and Satisfaction in Global Virtual Teams," *Academy of Management Learning & Education*, vol. 21, no. 2, pp. 236–260, Jun. 2022.

[5] V. C. Hughston, "An empirical study: Team charters and viability in freshmen engineering design," in *2013 IEEE Frontiers in Education Conference (FIE)*, Oct. 2013, pp. 629–631.

[6] N. Gitinabard, R. Okoilu, Y. Xu, S. Heckman, T. Barnes, and C. Lynch, "Student Teamwork on Programming Projects: What can GitHub logs show us?" Aug. 2020.

[7] A. f. C. M. Joint Task Force on Computing Curricula, IEEE Computer Society, "Software engineering 2014 curriculum guidelines for undergraduate degree programs in software engineering a volume of the computing curricula series," https://www.acm.org/binaries/content/assets/education/se2014.pdf, Feb 2015.

[8] S. Tenhunen, T. Männistö, M. Luukkainen, and P. Ihantola, "A systematic literature review of capstone courses in software engineering," 2023.

[9] K. Forsberg and H. Mooz, "The relationship of system engineering to the project cycle," in *Proceedings of the National Council for Systems Engineering First Annual Conference*, 1991, pp. 57–61.

[10] J. Vanhanen and T. O. A. Lehtinen, "Software Engineering Problems Encountered by Capstone Project Teams," *International Journal of Engineering Education*, 2014.

[11] C. J. Stettina, Z. Zhou, T. Bäck, and B. Katzy, "Academic education of software engineering practices: Towards planning and improving capstone courses based upon intensive coaching and team routines," in *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*, May 2013, pp. 169–178.

[12] M. C. Bastarrica, D. Perovich, and M. M. Samary, "What Can Students Get from a Software Engineering Capstone Course?" in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, May 2017, pp. 137–145.

[13] W. S. Smith and L. Lai, "A new requirements template for scientific computing," in *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, J. Ralyté, P. Ågerfalk, and N. Kraiem, Eds. Paris, France: In conjunction with 13th IEEE International Requirements Engineering Conference, 2005, pp. 107–121.

[14] B. Meyer, *Handbook of Requirements and Business Analysis*. Springer, 2022.

[15] S. Robertson and J. Robertson, *Mastering the Requirements Process*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co, 1999, ch. Volere Requirements Specification Template, pp. 353–391.

[16] G. Sudhakar, A. Farooq, and S. Patnaik, "Measuring Productivity of Software Development Teams," Rochester, NY, 2012.

[17] S. Saadat, O. B. Newton, G. Sukthankar, and S. M. Fiore, "Analyzing the Productivity of GitHub Teams based on Formation Phase Activity," in *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Dec. 2020, pp. 169–176.