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 among 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 index**

I. Introduction

The workload for a software engineering or computer science capstone team project is often unevenly distributed over time and among 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 among 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 think about why the problems exist. A student project is not the same environment as the workplace. Students often learn 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 s teammate.

To save student effort and to make expectations clear, 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. GitHub has been used successfully in teaching [1], [3], [4], [5], but we are not aware of literature suggesting GitHub templates for capstone courses. An advantage of templates is that they remove the paralyzation caused by too many choices when starting from scratch. With so many options, students can struggle with where to begin. The template makes standard infrastructure decisions for them, so that they can focus on their project.

An uneven workload distribution among 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 facilitating 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 unambiguously lays out their expectations. The idea of a team charter is not new [6], [7], [8], 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* [3], [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 (Section III). We then describe our proposed experiment for collecting more detailed data (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]. The course follows the ACM guidelines of 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 [9]. This course is currently delivered to 150 students divided into 29 groups of 4–5 members (the typical size for capstone courses [10]). 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, the team still explicitly identifies typical stakeholders/users for their project.

A. Structure and Timeline

Figure 1 show the V-model [11] 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. The Rev 0 documents are graded, but the emphasis is on formative assessment, so their weight is low. Documents are revised and re-graded at the end of the term (Rev1 Doc, Week 26). This second evaluation is based on how well the students incorporated feedback from the instructor, TAs and fellow students. The iteration allows students to produce a higher quality document for their summative review (Rev 1). An iterative process for a software capstone course is recommended by VanHanen and Lehtinen [12] to improve learning outcomes. Although there are reasonably frequent interactions with the TA and instructor, we do not follow the agile process recommended by some [13], [14].

In recognition of the value 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 the feasibility of the project, providing an opportunity to potentially revise the project scope. The Rev0 demo is expected to show off the final and complete product. The teams rarely achieve this goal, but the push for Rev0, together with the feedback they receive, 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.

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 teams spend building their project's infrastructure, and to standardize all the arbitrary decisions, like folder and document names. The standardization helps teams when doing peer reviews of each other's work and it improves communication between teams, teaching assistants and instructor. When

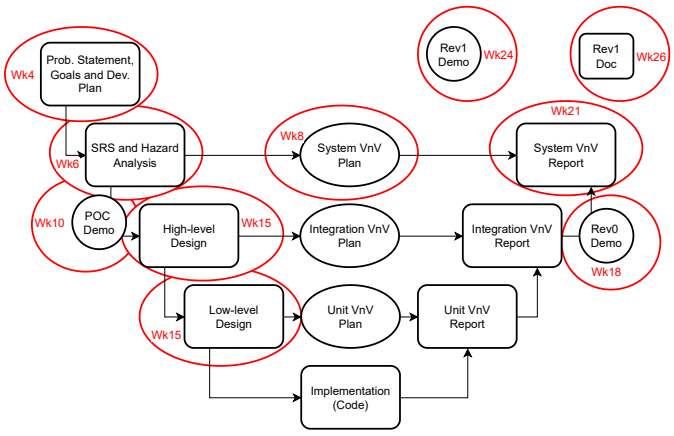

Fig. 1: V Model Used for Capstone Deliverables

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. Moreover, plain text documents are recommended for software engineering instructors working with GitHub [4] to take advantage of the diff and line commenting functionalities. 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 [15]), SRS-Meyer (a template by Bertrand Meyer [16]) and SRS-Volere (the Volere template [17]).

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. Teams are also encouraged to create their own issue templates, since empirical evidence shows that projects with templates exhibit reduced resolution time [18] and that templates are viewed positively by contributors and maintainers [19].

C. Team Engagement with Productivity Measures

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

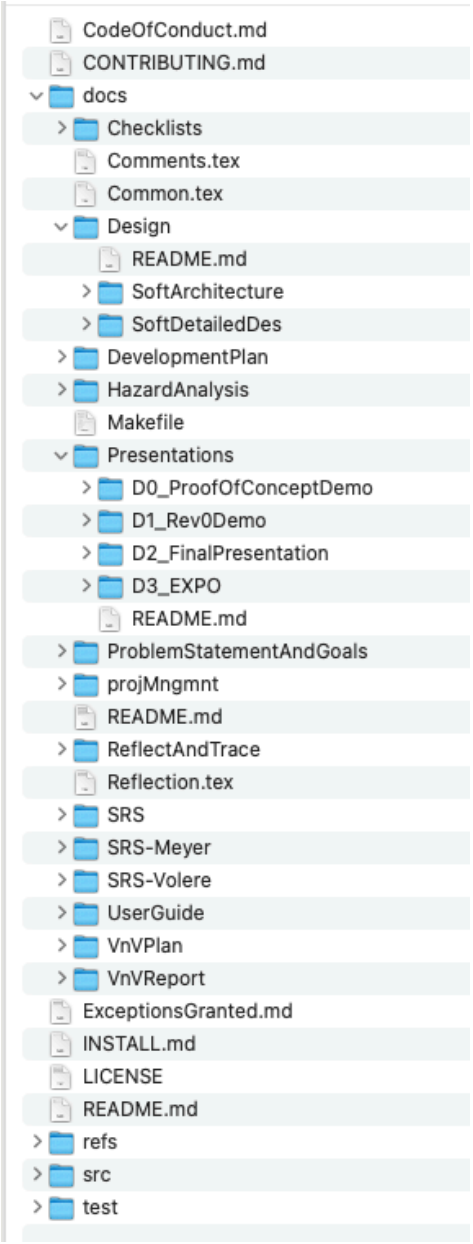

Fig. 2: GitHub Capstone Template

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 [20], but by default we keep things simple and standard for the teams. However, if a team desires more complex metrics, they can measure those alongside the required ones. The reality is that no single metric captures productivity [21], so we encourage collecting multiple metrics. 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 metrics. For instance, a team member may have focused their commits on a branch that has not yet been merged.

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 of *a priori* creation of rules is that the difficult discussion happens while relationships among 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 addressed, 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 grade can be multiplied by a "team contribution factor", if the data suggests this is necessary for fairness. The data for judging team contribution is not just commits. Any change in an individual's grade 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.

III. Preliminary Data

In this section we compare preliminary data for the 2022/23 and 2023/24 academic years. The two versions of the capstone course are similar. Both follow the V-model given in Section II-A and both use a Github template (Section II-B). The difference between them is that about a third of the way through the second course, teams were asked to begin measuring team performance metrics. Neither course included a team charter.

A. Time Spread

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.

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%)

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.

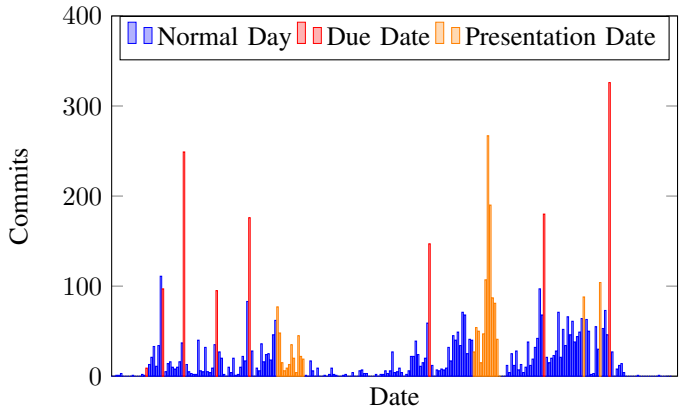

Fig. 3: Histogram of Commits for 2022–2023. Dates shown in red are due dates for major written deliverables, and dates in orange are days where presentations were scheduled.

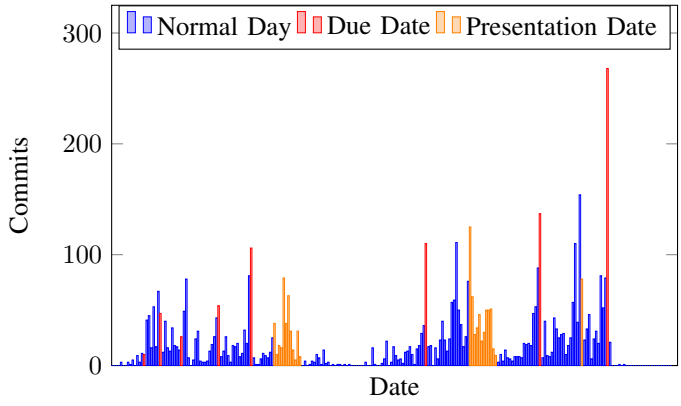

Fig. 4: Histogram of Commits for 2023–2024. Dates shown in red are due dates for major written deliverables, and dates in orange are days where presentations were scheduled.

B. Team Fairness

To measure the spread of work across team, one method is to compute a fairness index on the data, such as Jain's fairness index [22]. We instead developed an index tailored to our needs. We developed an index where values range from 0 to 1, and where all values in between contain meaningful information. We start with the following *unfairness* index:

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

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

The index 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* index, 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 index to other metrics (e.g. lines of code, issues created/closed, pull requests created/merged, etc.), and applying it to multiple metrics at once, similar to the multi-Jain fairness index proposed by some researchers (e.g. [23]). We would also like to investigate if there is a correlation between lower fairness values and perceived unfairness by the teammates themselves. Another research question would be if having live access to this fairness index encourages teams to share work more evenly or simply encourages them to "game the system" to increase the value artificially.

C. Time Fairness and Correlation to Team Fairness

We can apply the fairness index in III-A to the daily commit data from III-B. This results in a measure of how well-spread-out each team's work was across the course. In this case, a fairness value close to 0 indicates a high concentration of the workload on certain days, whereas a value of 1 would indicate evenly distributed work.

The results for each team are included in red in Figs. 5 & 6. As was the case in Section III-A, this index shows that work was not well-spread-out, with even the best teams having only 20% of their commits spread across time. In 2022/23, there was a weak negative Pearson correlation between team fairness

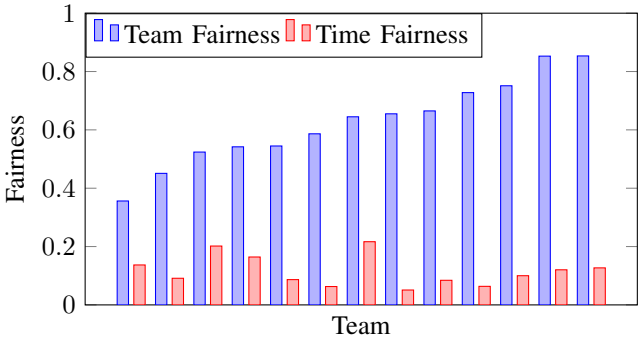

Fig. 5: Fairness of Commits Per Team 2022/23 [n=13; Team Fairness Mean: 0.63, Stddev: 0.15; Time Fairness Mean: 0.12, Stddev: 0.05; Correlation: -0.16]

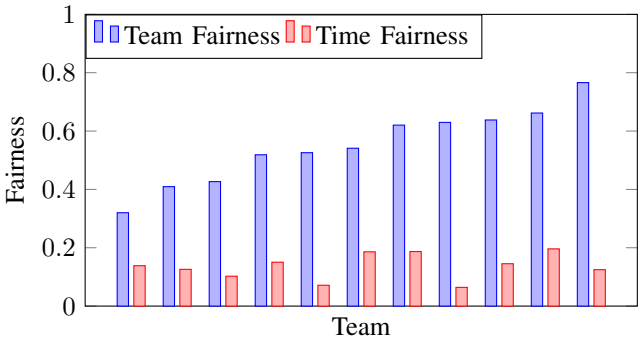

Fig. 6: Fairness of Commits Per Team 2023/24 [n=11; Team Fairness Mean: 0.55, Stddev: 0.13; Time Fairness Mean: 0.14, Stddev: 0.04; Correlation: 0.15]

and this time fairness index, and in 2023/24, a weak positive correlation. We can speculate that a negative correlation may indicate that teams have a few motivated individuals who do much of the work in a more time-spread-out way, but this should be studied more.

IV. Proposed Experiment

We believe that our rationale for the ideas of GitHub templates (Section II-B) and for team engagement with productivity measures (Section II-C), together with the above preliminary data, suggests that these ideas are worth further exploration. Below we describe a proposed experiment to collect experimental data and we highlight potential threats to validity.

A. Experiment

At our institution we have capstone courses in software engineering and computer science. The software engineering version is described in this paper, the computer science version does not have GitHub templates or teams explicitly measuring their productivity. Although not a perfect control, the computer science version can be compared to the next offering of the software engineering capstone. We would compare the courses using the data from the GitHub repositories for the time spread of work, team fairness and time fairness. Given that commits are not an ideal measure of productivity, because different people choose different points within their work to commit, going forward, we will investigate measuring productivity by a count of work events [24], where a work event is a push, merged pull request, issue comment or pull request. In addition to the quantified data from the repos, we will add entry and exit surveys for the students. The surveys will include a questionnaire and focus groups.

B. Threats to validity

We have identified the following threats to validity:

- As mentioned above, commits are not an ideal measure of productivity.
- The experiment is not fully controlled since two changes are being made to the course. The changes are made together because the productivity metrics would not be possible without a version control system. The focus group discussion should hopefully clarify how each change effects the students.
- We will be comparing different courses, with different instructors, and with students with different backgrounds. The focus group comments will be used to ascertain the importance of these differences.
- Generalizing the results from the study may be difficult if another capstone course follows a dramatically different structure from that described in Section II-A.

V. Concluding Remarks

We have presented two apparently new ideas for improving the distribution of work in a capstone project over time and among team members: 1) GitHub templates, and 2) team engagement via productivity measures. The templates save time for the teams, allow students to get started quickly and provide standardization that assists communication between teams, TAs and instructors. Using productivity metrics easily available from GitHub (like issues closed and commits), teams can monitor the progress of their team members. If desired, the template presented here could be forked and modified to match the requirements for a different capstone class. With the aid of a team charter teams can develop quantified rules for the consequences if a team member does not meet the team's agreed upon expectations.

Preliminary data shows that the problems of uneven distribution of work is real. Teams make more than a third of their commits within three days of their deadlines and the average fairness metric is around 0.6. Partial introduction of the interventions mentioned in this paper suggests they might be able to improve the situation. Therefore, we propose an experiment where a course that uses these interventions is compared to one that does not.

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] 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.

[4] J. Feliciano, M.-A. Storey, and A. Zagalsky, "Student experiences using GitHub in software engineering courses: A case study," in *Proceedings of the 38th International Conference on Software Engineering Companion*. Austin Texas: ACM, May 2016, pp. 422–431.

[5] Z. Xu, "Using Git to Manage Capstone Software Projects," in *Proceedings of ICCGI 2012 : The Seventh International Multi-Conference on Computing in the Global Information Technology*, 2012, pp. 1–6.

[6] 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.

[7] 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.

[8] 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.

[9] Joint Task Force on Computing Curricula, "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, IEEE Computer Society and Association for Computing Machinery, Tech. Rep., Feb 2015.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] E. Sülün, M. Saçakçı, and E. Tüzün, "An Empirical Analysis of Issue Templates Usage in Large-Scale Projects on GitHub," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–28, Jun. 2024.

[19] Z. Li, Y. Yu, T. Wang, Y. Lei, Y. Wang, and H. Wang, "To Follow or Not to Follow: Understanding Issue/Pull-Request Templates on GitHub," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2530–2544, Apr. 2023.

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

[21] C. Jaspan and C. Sadowski, *No Single Metric Captures Productivity*. Berkeley, CA: Apress, 2019, pp. 13–20. [Online]. Available: https://doi.org/10.1007/978-1-4842-4221-6_2

[22] R. K. Jain, D.-M. W. Chiu, W. R. Hawe *et al.*, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, vol. 21, p. 1, 1984.

[23] M. Köppen, K. Ohnishi, and M. Tsuru, "Multi-jain fairness index of per-entity allocation features for fair and efficient allocation of network resources," in *2013 5th International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2013, pp. 841–846.

[24] 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.