University of Glasgow | School of Computing Science

A Holistic Dashboard of Software Development Work-Time & Practices

Inesh Bose (2504266)

ABSTRACT

This project researches Developer Experience, task-effort estimations and the relationship between them; this is because failures & frustrations in software project comes from the lack of awareness of code with bad DX (as it is given low regard) and inaccurate resource allocation as there are no tools for developers to reflect on their work-time & practices. We provide a solution through the means of a centralised modular dashboard showing visualisations of their development activity as realistic breakdowns from the discrete events and objective metrics on their code repository. This was evaluated in realistic scenarios with 20 developers of various roles in different structures, as they work on their projects, to determine the usability and effect on their process, showing increased awareness of their practices, and realisation of establishing good DX in their codebase.

Concepts. *User experience; task estimations; version control*

Keywords. *Developer experience; developer productivity*

1. INTRODUCTION

There is significant importance given to a good User Experience in a product - which is reasonable - however, there is another area that requires equal importance which is Developer Experience; it pertains to the humans working on a project as software developers – but project frustration is widespread in any industry. A person's mood affects their problem-solving skills and productivity [5, 23, 39, 42], and in software development, this influences performance on certain tasks such as implementing, testing and debugging [27]. Companies and organisations take measures on helping their staff destress and have a better experience, but this may not be effective or formalised. Software is different to other industries as it requires a level of mental cognition in unique, non-repeating processes [52, 1, 26]. It must be realised that **humans are the most important aspect of software engineering [35]**, and we plan to address the common obstacles and frustrations they face as they apply various disciplines to deliver reliable, efficient and high-quality products [48].

User Experience (UX) involves all aspects of the interaction between **the end-user** & the product representing the *holistic* perspective of how a person feels using the system. There have been recognised heuristics outlined for interface design along with published articles in the International Organization for Standardization (ISO). ISO 9241[1] describes human-centered design as an approach to design systems that improve quality (of life) for users by 1) being comprehensive & reducing training, 2) reducing discomfort and stress, 3) contributing towards sustainable objects without threatening life, and most importantly 4) **increasing the productivity and enhancing effectiveness & efficiency of users and organisations** providing a competitive advantage.

Developer Experience (DX) is user experience from a developer's point-of-view, i.e. the user is a person involved in the process of software development, so the persona can be known to have relatively higher technical knowledge and the products they use are involved in the development process. It is part of a wider field called Developer Relations (DevRel) that focuses on managing relationships between organisations and developers; it can also be seen as part of marketing & sales than engineering being similar to the field of Public Relations (PR). DevRel only started to become mainstream over the last decade with increasing SaaS startups like Twilio, Zendesk and Intuit.

In the same decade, access to fast internet, performant browsers & mobile devices increased the demand for web development. PHP has been widely used to render HTML pages with CSS & JavaScript, but with the introduction of JavaScript runtimes such as Node.js and Deno, developers could just use the knowledge for JavaScript for the server and client. Frameworks like Express, Webpack and Bootstrap also provided the base implementation of a server, renderer and component styling (respectively) for faster development; being open-source, they relied on referrals (compounding growth), rather than funding (linear) marketing, so they focused on providing great developer experience, support and documentation for good retention.

For example, modern languages like Rust and TypeScript became popular due to the development experience. Stack Overflow Developer Survey 2022[2] reveals the most loved and most dreaded programming languages (see Figure 1); it also included a section on Developer Experience sharing that while more than 50% had CI/CD & DevOps functions, only 38% had a developer portal (for tools and services) for their organisation and only 16% had Innersource initiatives.

The COVID-19 pandemic saw software being released and updated in faster sprints; for example, video conferencing was necessary over lockdown, and Zoom & Microsoft Teams shipped with lots of bugs and minimum features that were fixed overtime with multiple releases. In such circumstances, companies aim to improve their resource allocation, process efficiency, and cost reduction, so they measure software productivity [19] using automated metrics like source lines of code (SLOC) written [57, 19], function points [2, 25], changes request fulfilled [14, 40], and vague *timesheets*; these are highly objective & inaccurate at capturing a developer's efforts. There could be a lot of effort put to make a commit of 3 additions and 2 deletions, and development should acknowledge it so future resources aren't wasted. Ensuring

[1] `https://www.iso.org/standard/77520.html`

[2] `https://survey.stackoverflow.co/2022/`

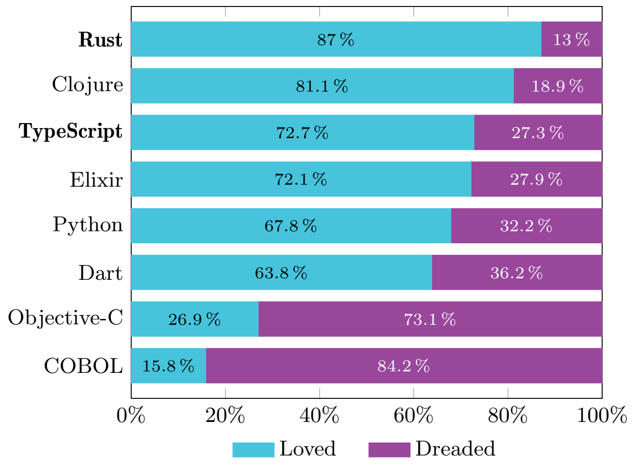


Figure 1: Responses for programming languages

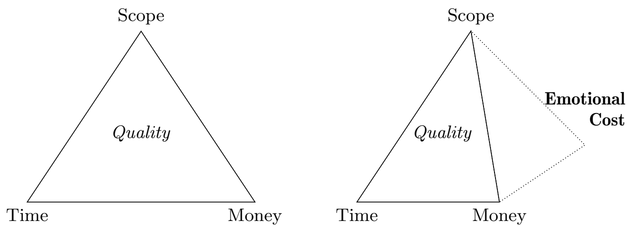


Figure 2: Project Management Triangle

good productivity in a process ensures good development experience [5, 23, 39, 42]. It is essential to recognise that developers are humans[3] and to include the hidden fourth-corner of The Project Management Triangle (Barners, 1980) which is "*Emotional Cost*" (see Figure 2).

Contributions. In this paper, we present the design and implementation of a dashboard that explicitly links developer experience with their productivity through the activity between the discrete commit events integrating data from sources and structuring it to produce new metrics that give a comprehensive insight into development effort so that task-estimations can be more accurate and developers increase productivity therefore improving their developer experience.

All information about the project can be found on `https://github.com/ineshbose/code-fitness`.

This paper is structured as follows. Section 2 describes previous work in related areas, highlighting the gaps and limitations of existing solutions. Section 3 presents our proposed solution and its design, explaining how it addresses the outlined issues. In Section 4, we describe the implementation details, including the tools and technologies used to develop the solution. Section 5 shares the evaluation methodology and results for our experiment, with feedback received from the users, showing how the solution performed in terms of usability, performance, and user satisfaction. Section 6 concludes the paper by summarising the key findings and contributions with potential future work that can build on this research to further enhance developer experience.

[3]`https://twitter.com/mahemoff/status/1947230411948032`

2. RELATED WORK

Major research in Developer Experience has not been established yet as it is a new area becoming mainstream only recently. Like user experience, DX is subjective but it is made up of objective attributes. We aimed to understand developers, their preferences and their different experiences by asking questions that can provide context on their background and characteristics.

How do developers learn new tools?

There are numerous tools available to develop software; this can be a new programming language or an external third-party library that provides some utility. A developer is different from a "non-developer" through knowledge of computing concepts and programming (writing code).

There are multiple ways to learn a new technology - there could be literature in form of a book or article, documentation hosted online, or a video tutorial. Producing these instructions (part of DevRel) involves human effort and requires the reader and writer to understand each other. Käfer et al. [28] compare (passive) text-based tutorials with video-based tutorials to understand which is better in terms of efficiency and learning; they discovered negligible differences for either approach despite each having their own sets of advantages and disadvantages [3]. Text tutorials were completed faster [38], while video tutorials helped faster learning over the comparatively longer period [32, 54] and were preferred by the participants when given the choice. It is useful for developers to have both video demonstrations and written literature, but some tutorials should require practical exercises to be completed by the readers so that documentation does not serve to "spoon-feed" developers.

Murphy-Hill et al. [41] look into "social solutions" that allow developers to learn new tools; these include peer observations, tool encounter, and RSS feeds. Pair programming is a popular practice in software development and it also enables a great amount of tool discovery between developers [16] as it opens opportunity to observe other developer's tools and have discussions. Furthermore, Begel and Simon [10] studies new recruits (novices) at Microsoft (a large organisation) to understand how they progress and transition to experts, finding that onboarding processes are tailored for this purpose and involve effective methods such as mentoring and pair-programming in the team. However, these rely on communication and the relationship between developers, and they may not share the same enthusiasm [56, 36].

Most importantly, searching on the internet is an important, common step while learning new tools. Bao et al. [8] found that there are more than 20 queries made by developers every day – a lot of it depends on the development phase, such as requirement analysis (where they search for explanations of terminologies, books and tutorials to understand business logic), the development phase (where they look for third-party libraries), or maintenance (looking for bugs that are commonly faced hopefully with a solution provided). Understanding queries also provides potential to understand developers' problems during the development process and possibly develop tools to help them. Bajracharya and Lopes [7], Stolee et al. [50] and Sadowski et al. [46] investigate how developers make search queries for their tasks, revealing different dimensions of search along with corresponding frequencies in order to highlight the importance of developing domain-specific search engines, automated generation & refinement of search queries, and knowledge sharing platforms; such solutions (like Stack Exchange) exist, but generating appropriate search queries (picking right keywords, adapting the traceback, etc) is an acquired development skill.

What do developers already know?

Experienced (senior) developers are aware of many tools, but they can still discover new ones [41], especially with new solutions being created at a fast rate with software. For awareness of their activity & practices, there are tools available to measure productivity, code health, test coverage and more; they work for projects most of the times, but they require high discipline and inference from the developers. There are also simple elements like autocomplete/IntelliSense and static analysers that provide feedback on code and developers must be able to diagnose and recover from the problems. Lajios [29] attempts to determine individual metric suites that fit into projects without correlating too highly with each other, for quality assessment & defect prediction, using machine learning and applying Correlation-based Feature Selection (CFS); they learnt that Weighted Methods per Class (Chidamber & Kemerer) and cyclomatic complexity (McCabe) were the most commonly suited metrics; these metrics may provide technical insight but they are not grounded to novice developers. Blondeau et al. [12] distinguishes metrics useful to predict the status of a project by conducting a literature review and evaluating a project at a major IT company; they learnt that metrics highly depend on the nature of each project as there are separate benchmarks & understandings of customer requests.

What circumstances are developers in?

Development is influenced by the structure of the process; it can involve working individually or in a team. Development experience & productivity is perceived differently for teams [45]. Tasks may depend on others' availability and learning could accelerate through pair programming [41]. Cooke et al. [17] and Lewis [31] view teams as a primary mechanism for leveraging specialised knowledge; this is why a software team is staffed according to the distribution of knowledge of tools required for successful completion of a project [58]; having only individuals with diverse knowledge is an insufficient condition to achieve quality performance in a software project [21]. Lack of trust, difficulties in communication, and lack of identification with project goals negatively impact the success of projects [60].

In a study by Mantei [34], group structures (outlined as Chief Programmer Team and Egoless Programming Team) were compared and classified their performance for managing software projects based on task properties such as difficulty, size, duration, modularity, reliability, time and sociability; it was revealed that a hybrid model (Controlled Decentralised) team is highly effective. Similarly, Sawyer [47] discusses the ways software developers organise information in archetypes of software development teams (Sequence, Group, and Network) discovering hybrid models to be good solutions. Other factors revealed by He et al. [24] include frequency of meetings as having positive effects on team cognition, emails having no effect, and gender diversity strongly affecting cognition positively.

Basili and Reiter Jr [9] studied relationships between the size of a programming group and several software metrics; however, modern agile teams are considered to must have less than 8 members. In evaluating team structures, Amrit et al. [6] sought coordination problems in teams using analysis tools and found clashes that were difficult to quantify accurately. López-Fernández et al. [33] studied teams in DevOps and classified them based on variables such as collaboration frequency, product ownership sharing, and autonomy, while Nan and Kumar [43] aimed to understand collaboration in open-source revealing that performance is directly proportional on size & structural interdependency. Clearly, there are many variables to development processes depending on the structure developers work in.

What motivates developers to be productive?

Research has been carried out to understand developer productivity and its influencing factors under different contexts; Trendowicz and Münch [52] lists these based on over a hundred publications along with their experiences involving industrial projects, workshops and surveys on software productivity, eventually learning that productivity of software development processes would depend significantly on the capabilities of the developers along with the tools and methods that they use; *the first step of quantifying productivity management is selecting the right metrics.*

More appropriate to large projects is to have experienced developers in their core team, hence Gill and Kemerer [22] looked at how productivity could be related to experience and found that they were directly proportional, i.e. software engineers with more experience were more productive; but using resources to recruit such engineers revealed declining returns; instead, putting greater emphasis on the design phase of a project is associated with increased productivity.

Lakhanpal [30] found team cohesion to be the dominating factor in team performance, but this was using a one-time survey. Canedo and Santos [13] revealed 37 factors to be pertinent to team productivity including collaboration [15, 49], team member availability [18, 37], and ease of communication [55, 59], while there's also individual characteristics such as work experience [18] and skill/competence [37, 44]. Ruvimova et al. [45] carried out a study in a practical environment with 25 individuals across five software teams reporting their perceived productivity for themselves and their team over interviews and surveys, and the most important finding from their study was realising that teams in modern organisations are highly fluid varying in size, members, structure and projects causing different aspects and dependencies to productivity. Similar to the previous question's study in this section, we realise that there are more complexities and steps in a developer's productivity.

How do developers manage their workload?

With respect to developers' personal lives and emotions, prioritising tasks is a crucial part of the process. Developers may switch between tasks based on their mental context or prioritise based on urgency. There have been established methods, such as Planning Poker, and software, such as Jira, to detail & organise tasks. Alhamed and Storer [4] describe limitations in effort estimation using planning poker, and use NLP models to predict estimates. Often, tasks can be seen as writing code (bugfix or new feature), tests or documentation. Developers tend to do these in a cyclic sequence – write code, write tests for it and then document the change; however switching between these may be difficult creating fatigue and developers tend to prefer working on one area, like writing code and no tests and documentation causing code-tasks to be prioritised. Moreover, Storer and Bob [51] point out frustrations in maintaining test cases in Behaviour Driven Development (BDD), and use NLP algorithms to create tests so that there is only one area of code to maintain for developers [11]. Along with that, information may not be central to Source Control Management (SCM) requiring more switching, so Edwards et al. [20] study and embed issues into the SCM to help reduce friction and the additional maintenance required by a developer. These have paved the way to develop solutions central to the SCM that help developers manage their tasks.

Summary – what did we find?

The field of software development has been extensively researched, and the literature has provided valuable insights into the various complexities involved in the process. One of the main conclusions drawn from this research is that software development is not a linear path; it is a highly iterative and constantly evolving process, with multiple influencing factors like team cohesion, learning curves, requirement comprehension, and communication. These factors can impact the speed and efficiency of the development process, therefore the quality of the final product.

However, the minor gaps we found in the literature come from the limited scope focusing only on a narrow range of contexts and failing to consider the wider ecosystem in which development processes take place. Developer Experience & Productivity has not been looked in both subjective and objective perspectives directly relating to developers' actions; there has also been relatively little focus on reducing friction in development, through tools or automation, that could significantly impact the dynamics of the process.

Our designed solution aims to solve this by providing a holistic view of developers' work-time, taking into account of their activity, efforts and the different contexts in the subjective developer experience, to provide more accurate and realistic measures for their productivity. This would enable us to gain a deeper understanding of the factors that influence software development and identify new ways to improve outcomes for both developers and organisations to ensure projects are completed on time and within budget.

3. DESIGN

As we have discussed the importance of User Experience and Human-Centered Design, we educate our implementation by outlining three fictional personas for our design.

- Developer 1: David is a 35-year-old senior software developer who graduated with a Bachelor's degree in Computing Science from the University of Edinburgh. He has been working in the industry for over 10 years, with experience in large organizations such as RBS, Skyscanner, and Amazon. He is now the tech lead for a SaaS startup based in Edinburgh. David is a perfectionist who values efficiency, scalability, and maintainability in software development. He has experience working with various programming languages such as Java, Python, and JavaScript and is comfortable working with both front-end and back-end technologies.

- Developer 2: Sarah is a 27-year-old junior software developer who graduated with a Master's degree in Graphic Design from the Glasgow School of Art. After graduation, she taught herself programming over the last two years by following Python tutorials & exercises on YouTube. Being a creative and visual thinker, Sarah enjoys designing user interfaces and user experiences (UI/UX), but she is new to the world of software development eager to learn more about programming languages and web development frameworks.

- Developer 3: Alex is a 28-year-old software developer who graduated from a programming boot camp three years ago. They enjoy solving complex problems with a keen interest in software development. They are proficient in developing features and documenting those features, making them an essential member of a SaaS startup team. Alex has experience working with React and Node.js and is eager to learn new technologies and frameworks to improve their skills further.

The three developers, having different backgrounds and skillsets, are working in the same SaaS startup as a team working to provide a website for the startup that displays their services, handles transactions and allows employees to write blog posts with SEO[4]. They must choose the right tools that will facilitate the entire development and ask questions like "*how fast will it enable us to create the MVP[5]?*", "*does the tool have documentation and integrate well into an ecosystem while keeping dependencies low?*" and "*will it ensure that the codebase is maintainable for the future (of the company)?*" - a lot of decisions tie to the experience developers would have working with the tool. They must also ensure that despite different skills and subjective DX, the development of the code itself is *objective*, i.e. it does not depend on who is writing the code and everything is consistent, so some rules must be chosen such as code style, directory structure, pipeline tests, automated hooks, and other similar factors. Code reviews also improve consistency but they are known to be big blockers in productivity and time-consuming [13] costing the company time and money.

A big part of the funding is used to compensate the developers (their pay), so they require detailed timesheets so that the data can be used to educate estimates and shared with investors for appropriate budgeting. The company could also use monitoring software, but this would be problematic – the software may not be specialised in studying developer activity requiring manual review and it could make employees conscious of their activity. The code repository shows activity, however, it only contains objective metrics and discrete events, such as commits, timestamps, issues closed, and SLOC, not covering the real effort of the developers when they're not writing code (reading documentation, doing code reviews, etc). Instead, the team can be provided with a hybrid monitoring tool that enables them to review their activity themselves, breaking down their efforts (see Figure 3), and use it during retrospective meetings.

Think of a scenario where the team were reported a bug on a page of their pilot prototype website with error-message "`TypeError: cannot read property of undefined (reading 'toString')`". This prototype uses a standard React front-end with a Django REST Framework back-end. To investigate this bug, there are lots of questions to be raised.

[4]Search Engine Optimisation
[5]Minimal Viable Product

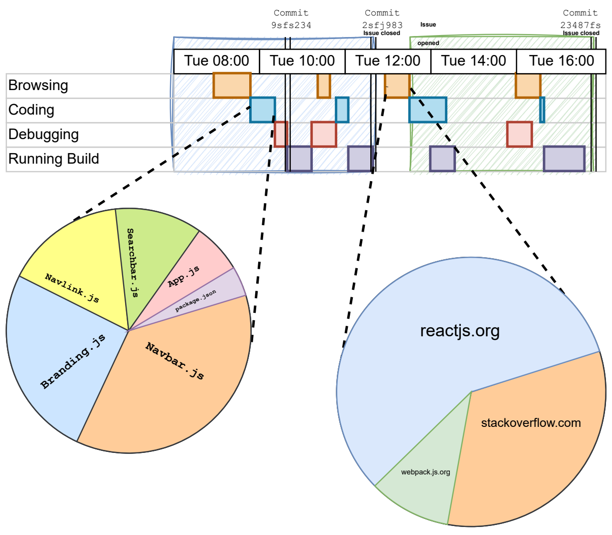


Figure 3: Example of efforts spent on a timeline

- ”*why was this not covered in tests?*”
- ”*what does the documentation say? what does Google say? what are the search keywords? will it trim down results and give an exact response?*”
- ”*why does it happen only one page? is it the routing library or the page component code?*”
- ”*is the code written in JavaScript or TypeScript? if TypeScript, the compiler should locate the issue*”
- ”*what does the traceback say? is the React app being served as a static asset through Django with Webpack building the compiled* `main.js` *in watch mode? if so, traceback is useless*”
- ”*if it is being served through Django, the file may also be cached in the browser and not reflect latest changes, so does cache need to be cleared every time (frustrating) or it needs to be tested on incognito?*”

The fix, carried out by Sarah over tasks outlined in Figure 4, just involved adding a condition to an `if` statement with commit diff as `+1 addition -0 deletions`. This may be a minor change, but the dashboard would verify Sarah's actual efforts that went into making this change creating a good point of conversation for the team in their retrospective so that they can plan a refactor and improve debugging for their project to provide better DX with faster fixes.

Application Design

Remote repository hosting services (such as GitHub, GitLab and Bitbucket) provide their unique set of features to enable developers to be able to progress on their development processes in a disciplined, streamlined manner if they prefer; this includes issue tracking, forking, wiki documentation, etc. While wiki documentation is designed as a separate repository with the same name as the main source code repository but with a `.wiki` suffix, many projects also tend to keep consistent documentation in a `docs/` directory that gets compiled and deployed to different sources. The repository README, however, has been a critical part to development; this acts as the home landing page to the project and developers tend to spend time in writing and/or understanding it. With the power of Markdown, it has become easy to write and preview the document. Moreover, Markdown Badges (or Shields) are a popular way to display information about the project such as license, code-size, pipeline status, out-of-date dependencies, etc [53] - this has made the README as a good source of all information required on the ”landing page”. This idea is the inspiration. Markdown is limited to the simple syntax; other services exist but they also have limitations such as being outside the SCM, limited metric options and unpredictable payment plans.

The proposed solution developed a dashboard that is integrated into the developers' workflow, through their Text Editor or IDE, and displays their efforts spent on completing tasks on their repository to ensure that they are aware of their precise development activity. The system uses a highly-integrated & modular architecture to enable quick and adaptive customisation of the dashboard if developers are only interested in certain metrics – similar to fitness applications like Google Fit and Fitbit that allow toggling of graphs and integration with third-party services, essentially making our tool to be a fitness app for software projects.

Visualisations

For the initial version of the dashboard, we needed to study and provide default visualisations that could interest developers and show potential for more metrics. We planned to display their sprint as a timeline. The information that developers mostly look for in their repositories are commits, issues, and pull/merge requests. We can use the time difference between two commits (to look into what they were working on), issue opening & closing (to look into how they were working), and pull request being created & merged/closed (to look into how they were reviewing) as a continuous event and display activity between those timestamps.

The visualisations must be unique from the ones that hosting services provide; they usually provide a contribution graph but they only consider commits while creating issues and pull requests is an equally important task. Teams may also want to see how their week was spent if sprints last over multiple weeks. Sunburst charts of their time would also provide useful, comprehensive breakdowns of their time. For example, say Alex spent 4 hours on closing an issue. 60% of the time was spent coding and within that 60%, 40% was spent on `AboutPage.js`. Finally, David may want to routinely review the repository file-by-file to see when the last changes to each file were made. If a file is changed frequently, that may not be optimised and would need to be broken down, or if a file is not changed frequently (say `package.json` or `requirements.txt`), it may be outdated. In this case, a form of a file-heatmap would be very useful.

4. IMPLEMENTATION

Similar to the situation described in Section 3 about selecting the right tools, we chose DX-focused tools, that integrate into an ecosystem, to develop the DX-based project.

The Editor

We're aware of the popularity and extensibility of the Visual Studio Code (VSCode) text editor; we created an extension that can be enabled within the editor to provide the dashboard within the interface. Since VSCode is based on

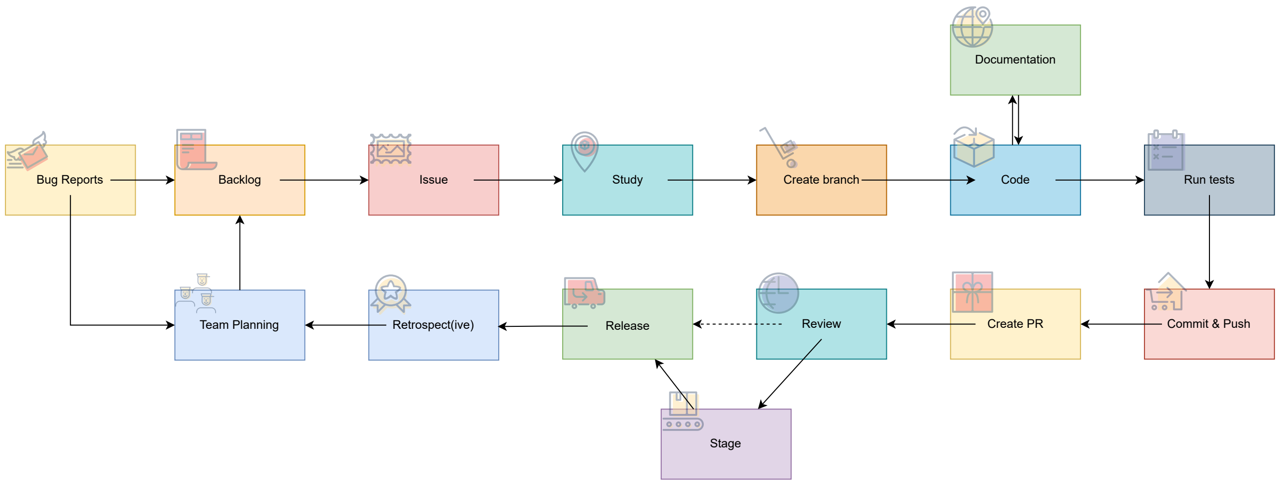


Figure 4: Workflow of a developer with an issue

the Electron framework, which renders Chromium browser instances, and is written in TypeScript, extensions would also need to be based on Node.js. Moreover, VSCode provides several APIs that can enable functionality for its extensions; the Webview API[6] allowed the extension to embed our interface into the editor similar to an `iframe`. The environment could not be assumed to be the same as a traditional web browser as VSCode resources are local and specific to the users' computers, so we needed to consider all restrictions. Our webview gets registered after VSCode calls the exported `activate` function, can be launched from its Command Palette and renders on the main panel that is used to view & edit code to take advantage of the large display with tabbed headings in groups. The extension was also published and listed on the Visual Studio Marketplace.

The Interface

While we were able to create a Visual Studio Code extension, the interface had to be developed separately; this would be similar to creating webpages with HTML, CSS & JavaScript. We chose to use SvelteKit[7], to develop the interface, which is a modern web framework based on Svelte & Vite (so DX was incredible) providing high performance & compilation with static site generation (SSG) enabled. Serving our compiled application through Visual Studio Code was a difficult task, as VSCode expects HTML templates to be provided as a string, but we developed a custom bundler, using HTML parsers and JavaScript code generators, to enable the integration and keep the UI easy to maintain.

Finally, the interface must provide visualisations as that is the core purpose of our dashboard, but generating graphs on websites requires `canvas` elements and complex JavaScript code that needs to handle input of unknown type. This step was crucial to ensure the modular architecture of our system, and we chose to use the Chart.js[8] library. It is the most popular JavaScript charting library (after D3.js) that can render different charts with great performance, responsiveness and customisation. The required input to generate a graph is a JSON object for which our system declared a consistent schema to be used in plugins.

The final product interface with generated graphs can be seen in Figure 5. Component styles remained consistent with VSCode with the help of the Webview UI Toolkit[9].

Modular Architecture

Using the core technologies mentioned above, we were able to setup our targeted modular architecture that could enable multiple visualisations to be generated from different

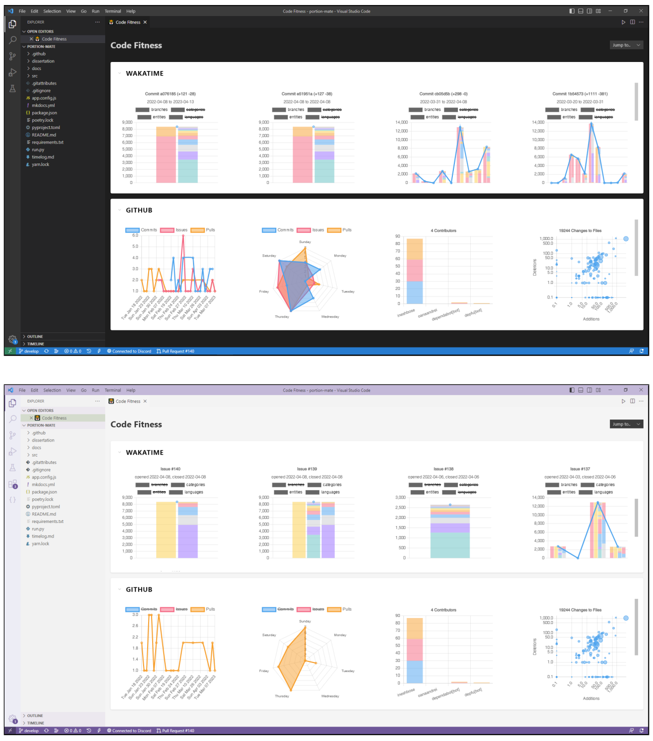


Figure 5: Dashboard Interface

[6] `https://code.visualstudio.com/api/extension-guides/webview`
[7] `https://kit.svelte.dev/`
[8] `https://www.chartjs.org/`
[9] `https://github.com/microsoft/vscode-webview-ui-toolkit`

sources without making changes to the extension or the interface. Since the repository uses the monorepo development strategy, each aspect of the system is abstracted into its own package, and integrated with the help of Turborepo and Yarn Workspaces. The `core` package implemented the logic of taking the configuration as input and loading it to process data from the plugins. Plugins are higher-order functions or objects with a `setup` function that provide access to APIs and return an `export` function back to be called by the loader; these can take declare required inputs from the configuration (such as endpoint URL, filter queries, and API token) and also take the loader context as input to get environment details and/or interact with other plugins; these functions can also be asynchronous with all `export` functions being executed concurrently. Since the project used TypeScript to ensure strong type-safety all-around the code, plugins take advantage of kit-utilities (such as `definePlugin`) exported by the core package to confirm that parameters and return values follow the schema. This architecture is heavily inspired by the Nuxt[10] framework.

The `extension` package uses the SvelteKit app that calls the loader with the configuration – it plugs into the extension (see Figure 6a) through the exported `webview` function generated by the custom builder. This custom builder mediates the Vite SSG template with hashed URLs for static assets so that the VSCode Webview can render properly with the `vscode://` protocol. This means that the extension is also designed to be framework-agnostic, i.e. it is not specific to our interface built through SvelteKit, but can be redeveloped in other front-end frameworks like React & Angular. In addition to the cyclic, asynchronous flow of control between the plugins & APIs in the core package (see Figure 6b), VSCode itself provides APIs and modularity through extensions, so our implementation also uses other VSCode extensions' utilities to generate data - like `vscode.git` to safely get remote URL, and `GitHub.vscode-pull-request-github` to automatically register API token. The `package.json` showcases usage of internal packages and configuration for VSCode through the "`contributes`" field.

Initial Plugins

As the plugins in the system provide Chart.js schema data for the visualisations, it would need to be generated from appropriate sources such as their repository (such as commits, issues), their IDE (keystrokes, build status) and their browser (websites, active tab). We developed plugins for GitHub as it is the most popular hosting platform and it provides a powerful API with detailed documentation. Some constraints were introduced due to the API requiring authentication and rate-limits; in most cases, we were able to adjust our design to be compliant and efficient.

The main plugin was for WakaTime[11]. It is a software productivity analytics service that collects and displays developers' coding activities through a range of integrations and plugins for popular development environments, including VSCode & Google Chrome, along with an API. This was extremely useful to the project as it removed massive complexities required for our system and our implementation would only need to map the data from the WakaTime API to the repository data generating suitable, intuitive graphs.

[10]`https://github.com/nuxt/nuxt`
[11]`https://wakatime.com/`

5. EVALUATION

We presented our developed dashboard to real developers, to evaluate its usability using formalised methods and determine its scope in their development process, by carrying out multiple experiments with them (and their teams) in unique circumstances to be able to understand subjective DX & productivity with a broad sample of participants.

Objectives

The main goal of our implementation was to provide developers with a visual representation of **realistic** measures that reflect their development activity; these are not limited like objective metrics but include other relevant factors that occur between the discrete commits (such as referring to documentation, writing tests, etc) to elaborate contexts for changes to the codebase. By presenting this information in a clear and concise manner, the dashboard aims to help developers evaluate their task estimation and effectiveness; it also can help identify bottlenecks & struggling points, which can then be addressed through refactoring and other improvements to enhance the developer experience.

The experiment sought to verify whether these objectives had been achieved by analysing the impact of the dashboard on developer productivity and satisfaction in estimating their future tasks and diagnosing blockers; it also looked to gather insight on the specific metrics that developers were interested and are hoping for.

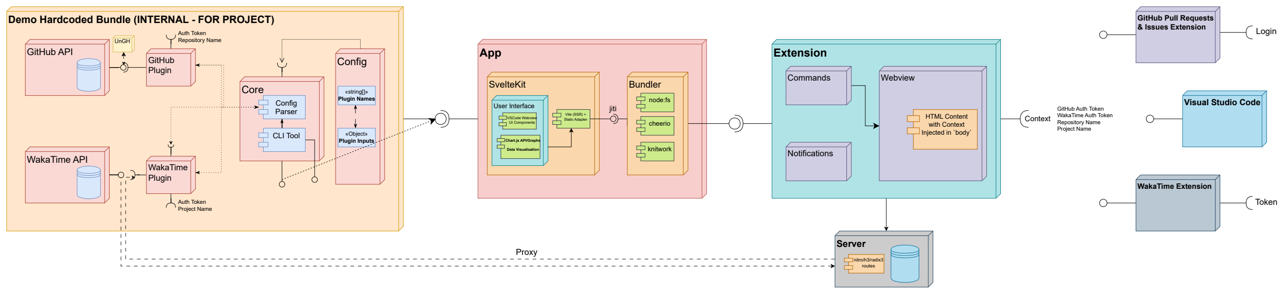
Demographics

We only recruited people that were software developers, familiar with issue tracking & version control and additionally working on a real software project (with/without a team) at the same time as the experiment. Aside from the survey we ran on online forums (through Reddit communities and Discord servers), all experiments were conducted in person at the University of Glasgow. The majority of participants were in their twenties, male and completing their Bachelors degree, while the sample also included developers with a Masters degree and/or who do not identify as male. 80% participants were involved with web development roles, 40% native apps development, 27% in DevOps, and 6% were involved in research. Despite all participants being exposed to version control before, about 85% of participants used it frequently; most used issue tracking and feature branching disciplines along with occasional coordination by communicating over Discord/Slack/Teams. Almost all of them used Visual Studio Code & GitHub, along with the usage of self-hosted instances of GitLab. Of participants that used the dashboard for their own projects, 10 were in an agile team who met regularly in person, 1 in a remote agile team and 1 working individually; most of these developers were aspirational recruits as they were learning professional software development through a course at the university. Most did not think much of their current productivity; see Figure 7.

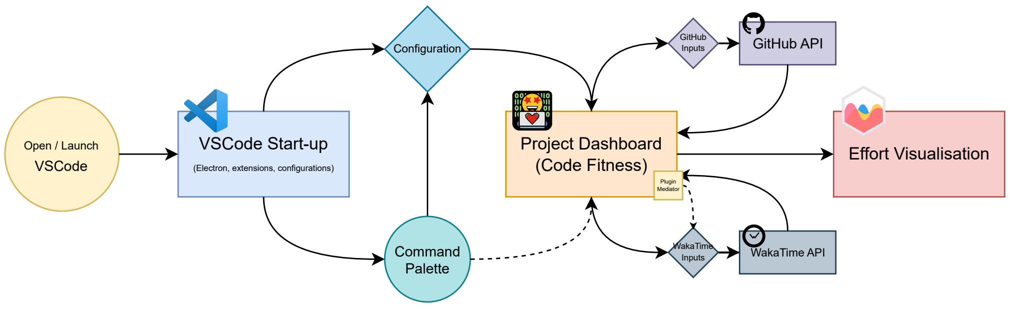
Methodology

Over the span on 11 weeks, we ran three experiments with the participants to gather results on the effectiveness of our dashboard. These were designed for different conditions on the evaluation nature and ran in parallel to each other.

The first experiment was a longitudinal study to be carried out for 6-8 weeks. It required developers to make use of the dashboard while they work on their projects over sprints

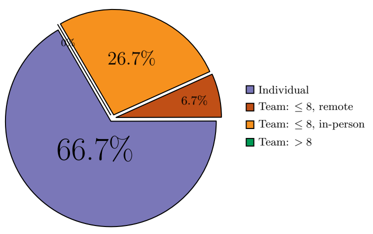


(a) Modular architecture of the system

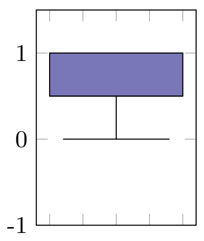


(b) Simplified user flow for using the dashboard

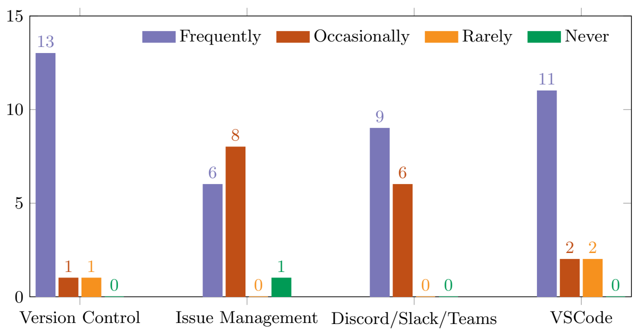
Figure 6: Infrastructure for the implementation



(a) Team Structure



(b) Project Satisfaction



(c) Tool usage

Figure 7: Demographic statistics of participants

and iterations; most worked in teams with other members also participating to generate more data about a team-based project. This experiment had three parts.

- *Introduction & Setup:* After recruitment, developers were provided with an overview of the dashboard and also assisted in getting setup with the extension on the devices they use for development.
- *Weekly Feedback:* As the participants worked on their software projects and there was more data available to display on the charts, they were asked to provide feedback on their experience using the dashboard in weekly checkpoints. The feedback form asked them to describe their week of working on their projects, if the dashboard had an effect on their week if there were metrics they were not expecting, and if they had any bug fixes or feature requests to report.
- *Retrospective:* At the end of the evaluation period, a retrospective meeting was conducted to study the dashboard with the data collected over the weeks and gather in-depth feedback from the participants on their overall experience with the dashboard.

The second experiment aimed to gather impressions and feedback on the dashboard using the industry standard System Usability Scale (SUS). This was designed as an alternative to the longitudinal study allowing developers to participate without their projects and with short commitment (at most 30 minutes). The tasks involved understanding a given project on a high-level and judging the resource allocation by visualising the efforts from the data presented by the dashboard. The sample project chosen for study, "*Portion Mate*[12]", was from our early projects for which we were

[12]`https://github.com/ineshbose/portion-mate`

already using the required technologies such as GitHub & WakaTime, with strict issue tracking and kanban boards, allowing sufficient data to be provided for the dashboard visualisations. The survey included an project exploratory pre-activity aimed to help participants get context and a sense of the project to understand the visualisations better.

The third experiment was an online survey helping our results to be accurate by reaching developers around the world. This survey asked developers about their perceptions of Developer Experience & Productivity by asking if they are conscious of project frustration, the metrics they use in their development, and the priority given in considering DX in their codebase. The survey only aimed to take 5-8 minutes to complete to be respectful of online participants' time, but they were also given the option to use the extension themselves for their projects and provide feedback.

Results

We were unable to gather formal results from the longitudinal evaluation and online survey as we faced time constraints and incomplete participation, however the usability survey reached completion and was able to provide insight. The responses from the ten SUS statements are based on a Likert scale – a five-point range from Strongly disagree (1) to Strongly agree (5) with 3 being neutral – and are used to determine a usability score for the system. The statements vary in describing the system as positive (denoted by +) or negative (denoted by -), so inverting the meaning of questions ensures that participants do not lose focus. The usability score is calculated using the following formula:

$$\left(\sum_{i=1}^{10} S_i\right) \times 2.5 \qquad S_i \begin{cases} r_i - 1 & \text{if positive } (i \;\%\; 2 > 0) \\ 5 - r_i & \text{if negative } (i \;\%\; 2 = 0) \end{cases}$$

where i is the statement number, S_i is a statement, and r_i is the scale position for the response to S_i. The scores can be seen in Table 1 along and the responses spread in Figure 8.

In addition, we conducted short verbal interviews with the developers involved in the longitudinal experiment to gather their feedback; transcriptions are listed in Table 2.

Discussion

Some experiments required quick adjustments or changes to be made last minute to the dashboard such as improving sanitisation for Git remote URL, readability in different VSCode themes, and integrating plugins in Unix environments. A key adjustment required to be made was developing a plugin that interacts with a private GitLab instance as participants' repositories were hosted there; this was developed in no more than two days using the same logic as the GitHub plugin and providing the same API required by the core module hence verifying that the modular architecture was achieved, and all revisions or additions to the code were building and deploying to the extension marketplace without issues - the implementation was successful.

Our main metric, the (system) usability score, came to be 73.8 with the corresponding grade as B- which means that the dashboard was an acceptable (to use) product; however, the score for each participant shows great variance (see Table 1 and fig. 9) ranging from 40.0 to 92.5, with a standard

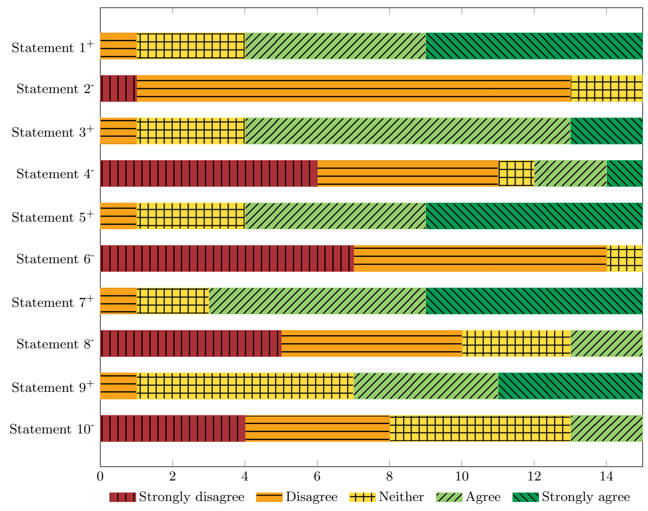


Figure 8: Responses for each statement

	Score	
Participant 1	65.0	C
Participant 2	92.5	A+
Participant 3	72.5	B-
Participant 4	40.0	F
Participant 5	80.0	A-
Participant 6	75.0	B
Participant 7	85.0	A+
Participant 8	72.5	B-
Participant 9	67.5	C
Participant 10	87.5	B+
Participant 11	60.0	D
Participant 12	70.0	C
Participant 13	72.5	B-
Participant 14	82.5	A
Participant 15	85.0	A+
Usability Score (Mean)	**73.83**	**B-**

Table 1: SUS Score for each participant

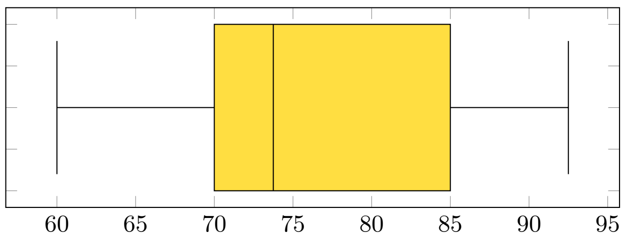
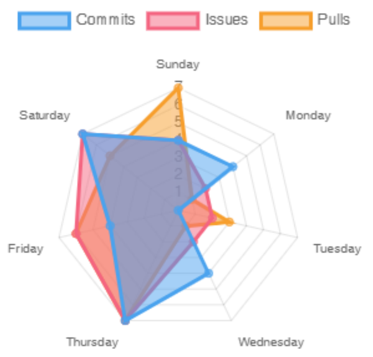


Figure 9: SUS Score Distribution



(a) Displayed graph for sample project

Question: How would you describe a week for this project?
Monday: Commits$^-$ (9); ”*catch-up*” (1)
Tuesday: Commits (2); Issues (1); Pulls$^-$ (3); ”*research*” (1)
Wednesday: Commits$^-$ (9); Issues (1); ”*work on issues*” (1)
Thursday: Commits$^+$ (13); Issues$^+$ (13); Pulls (6)
Friday: Commits (6); Issues (11); Pulls$^\sim$ (4)
Saturday: Commits$^+$ (12); Issues$^+$ (15); Pulls (4)
Sunday: Commits (5); Issues$^+$ (5); Pulls$^+$ (12)

(b) Grouped responses (out of 15)

Figure 10: Interpretation of the week-radar graph

Question: You were one of the developers who was asked to use the dashboard for a few weeks with your project. How would you say your experience was and influence on productivity?
It was useful in someways with the provided statistics.. not something I'd opening everyday, but once a week to help me remember. The data it provided for our project was pretty useful. If I were to begin again, I would be more efficient.
I didn't find it useful.. I would see GitLab which has nice charts there.. for estimations, we had things like planning poker and 5 whys.. I was already happy with my productivity.
Found it really helpful.. we spent a lot of time on documentation.. it would help my productivity if it was integrated with the project early on.
I find it interesting being able to see (more details about) the changes I made.. in general we're not good at remembering how long we spent on things.. if I was asked later and I didn't have the data in front of me, I wouldn't be sure.
Confusing at first, but a week later, it helped me personally, seeing the data.. I mainly use VSCode; I don't open GitLab, so its helpful than opening another tab for these metrics.

Table 2: Interview responses from developers

	Not Once	Just One	Two-three times
13/02 - 19/02	2	1	1
20/02 - 26/02	1	0	2
27/02 - 05/03	2	4	5
06/03 - 12/03	2	2	0
13/03 - 19/03	2	3	3

Table 3: Frequency of Dashboard Usage over weeks

deviation of 13, indicating that the user experience for developers using the system was very different and inconsistent. We explain this through the spread in Figure 8.

While 11 participants (73%) agree to thinking that they would use this system frequently (statement 1), 3 (20%) feel neutral or unclear about it and one participant (6.7%) disagrees; they shared that the remoting hosting service already provides graphs (second response in Table 2) despite using objective metrics. In comparison, these services have developed such graphs over many years and they do not plot large breakdowns on one chart which is efficient in rendering/interaction and understanding for users. However, no participant found the dashboard to be unnecessarily complex (statement 2) except for two participants (13.3%) who felt neutral about it; this may be because of their demographic that indicates occasional/rare use of version control and working individually more than in-teams.

There was still 73% agreement in the participants finding the system easy to use (statement 3), and the one disagreeing response uses version control less than frequently so the dashboard may have been difficult to understand in the context of the repository and revisions; in such cases, 20% of participants felt that they would need the support of a technical person to use the dashboard (statement 4). It can be noted that the metrics are new and overwhelming compared to traditional visualisations provided by current solutions; so we think a short tutorial may be helpful on the first launch showing the user where things are and how they can interact with data. 73% participants also agree to finding the various functions in the tool well-integrated (statement 5) with one participant disagreeing; this may be in terms of functionality, application or corresponding the right data, however participants did not see any inconsistency in the dashboard (statement 6); this may be biased since they were externally judging a sample project, but the responses also include participants from the longitudinal study.

There is a higher agreement among participants (80%) as being able to see the dashboard to be understood and used by most people very quickly (statement 7), but the benefit may not be seen by the remaining 20% as they, again, also do not frequently use version control so they would not have any need. While more than 50% participants did not find the system to very cumbersome to use (statement 8), there are about a third of the participants either felt neutral or indeed found it difficult to use; the reason for this is likely due to complex graphs, developed in the short-time, presented for an irrelevant project; this is evident in when 46.7% of participants did not feel very confident using the system (statement 9) as they are introduced to a new interface showing data for a new project that they did not work on and invest in personally to be able to judge the efforts. A similar 50/50 spread is with participants feeling like they needed to learn a lot of things to understand the dashboard (statement 10) with 53.4% participants disagreeing; this may also be biased as it includes participants from the longitudinal study who went had already gone through the learning curve, but on doing so, they realise the impact of the dashboard as they used it overtime (see all of Table 2, including the last response to demonstrate the learning curve).

We further verify developers' understanding of the visualisations through Figure 10 sharing an example of how developers interpreted the graph used to answer a required question in which they had to identify work tasks (commits, issues and PRs) distributed over the week. Almost all participants were able to identify that the majority of the work was done later in the week recognising Thursday as "the most active day". Some responses contained quantifying adjectives that were positive (like "commits were *mainly/mostly/especially frequent* on Saturday", denoted by +), negative ("*some/somewhat/few* pulls on Tuesday", denoted by -), and uncertain ("pulls *potentially* on Friday", denoted by ∼). A participant also inferred that the developers' week "*start with a catch-up, setup and research*" on the project; this demonstrates that developers could pick up information differently, some more than others. Similarly, we asked the developers to provide speculation on a large PR of 142 commits being made in a short-time based on the metric by the dashboard; after investigating through the breakdown, they were able to spot that all previous changes were made on the `develop` branch and finally being merged for release.

During our longitudinal study, we asked developers at the end of every week using the dashboard if they sought to introduce changes in their process after reflecting on their efforts for the week. Most responses did not share any intentions mostly because 1) there wasn't enough activity data at first, 2) the generated data wasn't anything that they were profoundly unexpecting, and 3) their projects were at the last iteration so changes would be too late, but some developers were able to notice the quality of their commits and inconsistency of work. Rather, they were more motivated to break down commits further and be pushing changes more frequently. We see through the positive responses in Table 2 and the distribution of participants using the dashboard each week Table 3 (regardless of participant drop or spike) that some developers picked up on the importance and the applications of such metrics opening the dashboard more than two times during the week (rather than everyday as pointed by the first response in Table 2). This reveals the overall themes from our result as being able to 1) give an overview of how developers and teams are performing with insight on that behaviour, 2) increase commit frequency, and most importantly, 3) enable self-awareness.

We point out and acknowledge the limitations of our experiments in generating significant results; these include factors such as short timeframe, convenience sampling, and rushed implementation of our tool. We address these further in the next section with plans for improvements.

6. CONCLUSIONS

This paper has discussed the importance of Developer Experience in relation to productivity and its impact on the success of projects. We presented a new tool that exposes developers' activities carried out for completing tasks on their repository to provide more accurate and realistic measures of productivity. The initial evaluation of this tool was carried out with 20 developers of different demographics such as roles, disciplines and team structures; it reports that developers were able to gain more detail to each revision in a repository and realise the time spent on it, helping them improve their practices such as committing frequently and consistently. The usability score for our system (73.8) in such early stages shows promise and scope for the project, but not all developers were able to perceive the metrics, thus not seeking to use it in the current stage.

In regard to the limitations of our study, first, the time constraints included implementing our tool, recruiting participants and running longitudinal evaluations; this didn't allow much flexibility and left little time for each phase to either be rushed or cut down. Second, due to the tight schedule, it was convenient to recruit friends & colleagues at the University of Glasgow for our experiments; this opened the potential for bias in the results as they may have provided diplomatic and socially desirable responses to the questions. Moreover, this sample of 20 participants was small and not representative of all developers as they were all novice graduate students at the university with relatively limited knowledge and experience in industrial software engineering. When the implementation was finished and ready to be evaluated, the developers had already crossed the research & learning phase of their projects and were on the final iteration. Finally, third, the implemented tool, only developed in a few weeks, did not give a seamless experience to the users and had limitations in the data such as accuracy, proper detailing and graph issues (such as axis, toggles, and size).

Developer Experience is a new & highly subjective field; any research in understanding this area will benefit software projects everywhere. More plugins and longer structured experiments for the system can provide more insight. The evaluation of our tool brought out useful feedback from the developers such as providing integration with other IDEs or repository wikis, alerting developers on new activities, and storing diary entries in a new tab. The idea of the dashboard was to help developers and their teams make accurate estimates for future tasks by basing their considerations on their experience & metrics for the previous ones, but it was up to the developers to infer the data. As also noted by a participant's feedback, there is potential for the system to specialise in providing estimation facilities such as taking story-points as input and verifying that effort metrics correspond to it or predict otherwise using algorithms. Moreover, investigating work-time practices to generate visualisations has made us realise the large amount of context switching that developers require, affecting the mental load and ultimately the DX. Our dashboard integrated into VSCode because we mentioned the importance of having information central to the SCM [20]; but the implementation could help promote more information to the central dashboard. There is also scope for relating DX & Productivity with emerging tools like ChatGPT and GitHub Copilot through the dashboard. Overall, this study has opened up new avenues for future research in the field of Developer Experience.

Acknowledgments. Thanks to everyone and every organisation, especially the JavaScript/Vue/Nuxt community, that has been a part of my Computing Science journey academically and professionally helping me enjoy creating software with no pain for users and developers equally. I'm grateful and appreciative of Dr. Tim Storer for supervising me on this project and being understanding from both research-perspective and engineering-perspective. Finally, I would also like to thank **you** for giving your valuable time to read through this paper - it means the world. Thank you.

REFERENCES

[1] T. Abdel-Hamid. The slippery path to productivity improvement. *IEEE Software*, 13(4):43–52, July 1996. ISSN 1937-4194. doi: 10.1109/52.526831.

[2] A. J. Albrecht. Measuring application development productivity. In *Proc. Joint share, guide, and IBM application development symposium, 1979*, 1979.

[3] K. P. Alexander. The Usability of Print and Online Video Instructions. *Technical Communication Quarterly*, 22(3):237–259, July 2013. ISSN 1057-2252. doi: 10.1080/10572252.2013.775628. URL `https://doi.org/10.1080/10572252.2013.775628`. Publisher: Routledge.

[4] M. Alhamed and T. Storer. Evaluation of Context-Aware Language Models and Experts for Effort Estimation of Software Maintenance Issues. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 129–138, Oct. 2022. doi: 10.1109/ICSME55016.2022.00020. ISSN: 2576-3148.

[5] T. Amabile and S. Kramer. *The Progress Principle: Using Small Wins to Ignite Joy, Engagement, and Creativity at Work*. Harvard Business Press, 2011. ISBN 978-1-4221-9857-5. Google-Books-ID: 7YgGqk3pD3oC.

[6] C. Amrit, J. van Hillegersberg, and K. Kumar. Identifying Coordination Problems in Software Development: Finding Mismatches between Software and Project Team Structures, Jan. 2012. URL `http://arxiv.org/abs/1201.4142`. arXiv:1201.4142 [cs].

[7] S. K. Bajracharya and C. V. Lopes. Analyzing and mining a code search engine usage log. *Empirical Software Engineering*, 17(4):424–466, Aug. 2012. ISSN 1573-7616. doi: 10.1007/s10664-010-9144-6. URL `https://doi.org/10.1007/s10664-010-9144-6`.

[8] L. Bao, Z. Xing, X. Wang, and B. Zhou. Tracking and Analyzing Cross-Cutting Activities in Developers' Daily Work (N). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 277–282, Nov. 2015. doi: 10.1109/ASE.2015.43.

[9] V. R. Basili and R. W. Reiter Jr. An investigation of human factors in software development. *Computer*, 12 (12):21–38, 1979.

[10] A. Begel and B. Simon. Novice software developers, all over again. In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 3–14, New York, NY, USA, Sept. 2008. Association for Computing Machinery. ISBN 978-1-60558-216-0. doi: 10.1145/1404520.1404522. URL `https://doi.org/10.1145/1404520.1404522`.

[11] L. P. Binamungu, S. M. Embury, and N. Konstantinou. Maintaining behaviour driven development specifications: Challenges and opportunities. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 175–184, Mar. 2018. doi: 10.1109/SANER.2018.8330207.

[12] V. Blondeau, N. Anquetil, S. Ducasse, S. Cresson, and P. Croisy. Software metrics to predict the health of a project? An assessment in a major IT company. In *Proceedings of the International Workshop on Smalltalk Technologies*, IWST '15, pages 1–8, New York, NY, USA, July 2015. Association for Computing Machinery. ISBN 978-1-4503-3857-8. doi: 10.1145/2811237.2811294. URL `https://doi.org/10.1145/2811237.2811294`.

[13] E. D. Canedo and G. A. Santos. Factors Affecting Software Development Productivity: An empirical study. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES '19, pages 307–316, New York, NY, USA, Sept. 2019. Association for Computing Machinery. ISBN 978-1-4503-7651-8. doi: 10.1145/3350768.3352491. URL `https://doi.org/10.1145/3350768.3352491`.

[14] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 2–11, New York, NY, USA, Oct. 2008. Association for Computing Machinery. ISBN 978-1-59593-971-5. doi: 10.1145/1414004.1414008. URL `https://doi.org/10.1145/1414004.1414008`.

[15] V. A. Clincy. Software development productivity and cycle time reduction. *Journal of Computing Sciences in Colleges*, 19(2):278–287, Dec. 2003. ISSN 1937-4771.

[16] A. Cockburn and L. Williams. The costs and benefits of pair programming. In *Extreme programming examined*, pages 223–243. Addison-Wesley Longman Publishing Co., Inc., USA, May 2001. ISBN 978-0-201-71040-3.

[17] N. J. Cooke, P. A. Kiekel, and E. E. Helm. Measuring Team Knowledge During Skill Acquisition of a Complex Task. *International Journal of Cognitive Ergonomics*, 5(3):297–315, Sept. 2001. ISSN 1088-6362. doi: 10.1207/S15327566IJCE0503_10. URL `https://doi.org/10.1207/S15327566IJCE0503_10`.

[18] C. De O. Melo, D. S. Cruzes, F. Kon, and R. Conradi. Interpretative case studies on agile team productivity and management. *Information and Software Technology*, 55(2):412–427, Feb. 2013. ISSN 0950-5849. doi: 10.1016/j.infsof.2012.09.004. URL `https://doi.org/10.1016/j.infsof.2012.09.004`.

[19] P. Devanbu, S. Karstu, W. Melo, and W. Thomas. Analytical and empirical evaluation of software reuse metrics. In *Proceedings of the 18th international conference on Software engineering*, ICSE '96, pages 189–199, USA, May 1996. IEEE Computer Society. ISBN 978-0-

8186-7246-0.

[20] N. Edwards, D. Jongsuebchoke, and T. Storer. Sciit: Embedding issue tracking in source control management. *Science of Computer Programming*, 206:102628, June 2021. ISSN 0167-6423. doi: 10.1016/j.scico.2021.102628. URL `https://www.sciencedirect.com/science/article/pii/S0167642321000216`.

[21] S. Faraj and L. Sproull. Coordinating Expertise in Software Development Teams. *Management Science*, 46(12):1554–1568, Dec. 2000. ISSN 0025-1909. doi: 10.1287/mnsc.46.12.1554.12072. URL `https://pubsonline.informs.org/doi/abs/10.1287/mnsc.46.12.1554.12072`. Publisher: INFORMS.

[22] G. K. Gill and C. F. Kemerer. Productivity impacts of software complexity and developer experience. 1990. Publisher: Cambridge, Mass.: Sloan School of Management, Massachusetts Institute of . . . .

[23] D. Graziotin, X. Wang, and P. Abrahamsson. Do feelings matter? On the correlation of affects and the self-assessed productivity in software engineering. *Journal of Software: Evolution and Process*, 27(7):467–487, July 2015. ISSN 2047-7473. doi: 10.1002/smr.1673. URL `https://doi.org/10.1002/smr.1673`.

[24] J. He, B. S. Butler, and W. R. King. Team Cognition: Development and Evolution in Software Project Teams. *Journal of Management Information Systems*, 24(2):261–292, Oct. 2007. ISSN 0742-1222. doi: 10.2753/MIS0742-1222240210. URL `https://doi.org/10.2753/MIS0742-1222240210`. Publisher: Routledge.

[25] C. Jones. Software Metrics: Good, Bad and Missing. *Computer*, 27(9):98–100, Sept. 1994. ISSN 0018-9162. doi: 10.1109/2.312055. URL `https://doi.org/10.1109/2.312055`.

[26] C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30 (5):416–429, May 1987. ISSN 0001-0782. doi: 10.1145/22899.22906. URL `https://doi.org/10.1145/22899.22906`.

[27] I. A. Khan, W.-P. Brinkman, and R. M. Hierons. Do moods affect programmers' debug performance? *Cognition, Technology and Work*, 13(4):245–258, Nov. 2011. ISSN 1435-5558. doi: 10.1007/s10111-010-0164-1. URL `https://doi.org/10.1007/s10111-010-0164-1`.

[28] V. Käfer, D. Kulesz, and S. Wagner. What is the best way for developers to learn new software tools? An empirical comparison between a text and a video tutorial. *arXiv preprint arXiv:1704.00074*, 2017.

[29] G. Lajios. Software metrics suites for project landscapes. In *2009 13th european conference on software maintenance and reengineering*, pages 317–318, 2009. doi: 10.1109/CSMR.2009.22.

[30] B. Lakhanpal. Understanding the factors influencing the performance of software development groups: An exploratory group-level analysis. *Information and Software Technology*, 35(8):468–473, Aug. 1993. ISSN 0950-5849. doi: 10.1016/0950-5849(93)90044-4. URL `https://www.sciencedirect.com/science/article/pii/095058499390044A`.

[31] K. Lewis. Measuring transactive memory systems in the field: Scale development and validation. *Journal of Applied Psychology*, 88:587–604, 2003. ISSN 1939-1854. doi: 10.1037/0021-9010.88.4.587. Place: US Publisher: American Psychological Association.

[32] S. A. Lloyd and C. L. Robertson. Screencast Tutorials Enhance Student Learning of Statistics. *Teaching of Psychology*, 39(1):67–71, Jan. 2012. ISSN 0098-6283. doi: 10.1177/0098628311430640. URL `https://doi.org/10.1177/0098628311430640`.

[33] D. López-Fernández, J. Díaz, J. García, J. Pérez, and A. González-Prieto. DevOps Team Structures: Characterization and Implications. *IEEE Transactions on Software Engineering*, 48(10):3716–3736, Oct. 2022. ISSN 1939-3520. doi: 10.1109/TSE.2021.3102982.

[34] M. Mantei. The effect of programming team structures on programming tasks. *Communications of the ACM*, 24(3):106–113, Mar. 1981. ISSN 0001-0782. doi: 10.1145/358568.358571. URL `https://doi.org/10.1145/358568.358571`.

[35] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices.* Prentice Hall PTR, USA, 2003. ISBN 978-0-13-597444-5.

[36] P. Marzolo, M. Guazzaloca, and P. Ciancarini. “Extreme Development” as a Means for Learning Agile. In G. Succi, P. Ciancarini, and A. Kruglov, editors, *Frontiers in Software Engineering*, Communications in Computer and Information Science, pages 158–175, Cham, 2021. Springer International Publishing. ISBN 978-3-030-93135-3. doi: 10.1007/978-3-030-93135-3_11.

[37] K. D. Maxwell and P. Forselius. Benchmarking Software-Development Productivity. *IEEE Software*, 17 (1):80–88, Jan. 2000. ISSN 0740-7459. doi: 10.1109/52.820015. URL `https://doi.org/10.1109/52.820015`.

[38] L. S. Mestre. Student preference for tutorial design: a usability study. *Reference Services Review*, 40(2):258–276, Jan. 2012. ISSN 0090-7324. doi: 10.1108/00907321211228318. URL `https://doi.org/10.1108/00907321211228318`.

[39] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software Developers' Perceptions of Productivity. Nov. 2014. doi: 10.5167/UZH-98324. URL `https://www.zora.uzh.ch/id/eprint/98324`. Publisher: s.n.

[40] C. Miller, P. Rodeghero, M.-A. Storey, D. Ford, and T. Zimmermann. ”How Was Your Weekend?” Software Development Teams Working From Home During COVID-19. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 624–636, May 2021. doi: 10.1109/ICSE43902.2021.00064. ISSN: 1558-1225.

[41] E. Murphy-Hill, D. Y. Lee, G. C. Murphy, and J. Mcgrenere. How Do Users Discover New Tools in Software Development and Beyond? *Computer Supported Cooperative Work*, 24(5):389–422, Oct. 2015. ISSN 0925-9724. doi: 10.1007/s10606-015-9230-9. URL `https://doi.org/10.1007/s10606-015-9230-9`.

[42] S. C. Müller and T. Fritz. Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and

Progress. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 688–699, May 2015. doi: 10.1109/ICSE.2015.334. ISSN: 1558-1225.

[43] N. Nan and S. Kumar. Joint Effect of Team Structure and Software Architecture in Open Source Software Development. *IEEE Transactions on Engineering Management*, 60(3):592–603, Aug. 2013. ISSN 1558-0040. doi: 10.1109/TEM.2012.2232930. Conference Name: IEEE Transactions on Engineering Management.

[44] E. Oliveira, T. Conte, M. Cristo, and E. Mendes. Software Project Managers' Perceptions of Productivity Factors: Findings from a Qualitative Study. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, pages 1–6, New York, NY, USA, Sept. 2016. Association for Computing Machinery. ISBN 978-1-4503-4427-2. doi: 10.1145/2961111.2962626. URL `https://doi.org/10.1145/2961111.2962626`.

[45] A. Ruvimova, A. Lill, J. Gugler, L. Howe, E. Huang, G. Murphy, and T. Fritz. An exploratory study of productivity perceptions in software teams. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, pages 99–111, New York, NY, USA, July 2022. Association for Computing Machinery. ISBN 978-1-4503-9221-1. doi: 10.1145/3510003.3510081. URL `https://doi.org/10.1145/3510003.3510081`.

[46] C. Sadowski, K. T. Stolee, and S. Elbaum. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 191–201, New York, NY, USA, Aug. 2015. Association for Computing Machinery. ISBN 978-1-4503-3675-8. doi: 10.1145/2786805.2786855. URL `https://doi.org/10.1145/2786805.2786855`.

[47] S. Sawyer. Software development teams. *Communications of the ACM*, 47(12):95–99, Dec. 2004. ISSN 0001-0782. doi: 10.1145/1035134.1035140. URL `https://doi.org/10.1145/1035134.1035140`.

[48] I. Sommerville. *Software Engineering*. Addison-Wesley, 1992. ISBN 978-0-201-56529-4. Google-Books-ID: DoJQAAAAMAAJ.

[49] C. Stephanidis. *HCI International 2015 - Posters' Extended Abstracts: International Conference, HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015. Proceedings, Part II.* Springer, July 2015. ISBN 978-3-319-21383-5. Google-Books-ID: 9Lg0CgAAQBAJ.

[50] K. T. Stolee, S. Elbaum, and D. Dobos. Solving the Search for Source Code. *ACM Transactions on Software Engineering and Methodology*, 23(3):26:1–26:45, June 2014. ISSN 1049-331X. doi: 10.1145/2581377. URL `https://doi.org/10.1145/2581377`.

[51] T. Storer and R. Bob. Behave Nicely! Automatic Generation of Code for Behaviour Driven Development Test Suites. In *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 228–237, Sept. 2019. doi: 10.1109/SCAM.2019.00033. ISSN: 2470-6892.

[52] A. Trendowicz and J. Münch. Chapter 6 Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences. In *Advances in Computers*, volume 77, pages 185–241. Elsevier, Jan. 2009. doi: 10.1016/S0065-2458(09)01206-6. URL `https://www.sciencedirect.com/science/article/pii/S0065245809012066`.

[53] A. Trockman, S. Zhou, C. Kästner, and B. Vasilescu. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 511–522, New York, NY, USA, May 2018. Association for Computing Machinery. ISBN 978-1-4503-5638-1. doi: 10.1145/3180155.3180209. URL `https://doi.org/10.1145/3180155.3180209`.

[54] H. van der Meij and J. van der Meij. A comparison of paper-based and video tutorials for software learning. *Computers & Education*, 78:150–159, Sept. 2014. ISSN 0360-1315. doi: 10.1016/j.compedu.2014.06.003. URL `https://www.sciencedirect.com/science/article/pii/S0360131514001353`.

[55] S. Wagner and M. Ruhe. A Systematic Review of Productivity Factors in Software Development, Jan. 2018. URL `http://arxiv.org/abs/1801.06475`. arXiv:1801.06475 [cs].

[56] W. M. Waite, M. H. Jackson, A. Diwan, and P. M. Leonardi. Student culture vs group work in computer science. *ACM SIGCSE Bulletin*, 36(1):12–16, Mar. 2004. ISSN 0097-8418. doi: 10.1145/1028174.971308. URL `https://doi.org/10.1145/1028174.971308`.

[57] C. E. Walston and C. P. Felix. A method of programming measurement and estimation. *IBM Systems Journal*, 16(1):54–73, Mar. 1977. ISSN 0018-8670. doi: 10.1147/sj.161.0054. URL `https://doi.org/10.1147/sj.161.0054`.

[58] D. B. Walz, J. J. Elam, and B. Curtis. Inside a software design team: knowledge acquisition, sharing, and integration. *Communications of the ACM*, 36(10):63–77, Oct. 1993. ISSN 0001-0782. doi: 10.1145/163430.163447. URL `https://doi.org/10.1145/163430.163447`.

[59] M. Yilmaz, R. V. O'Connor, and P. Clarke. Effective Social Productivity Measurements during Software Development — An Empirical Study. *International Journal of Software Engineering and Knowledge Engineering*, 26(03):457–490, Apr. 2016. ISSN 0218-1940. doi: 10.1142/S0218194016500194. URL `https://www.worldscientific.com/doi/abs/10.1142/S0218194016500194`.

[60] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt. Empirical evidence in global software engineering: a systematic review. *Empirical Software Engineering*, 15(1): 91–118, Feb. 2010. ISSN 1382-3256. doi: 10.1007/s10664-009-9123-y. URL `https://doi.org/10.1007/s10664-009-9123-y`.