User-Driven Technical Development of Design Systems

Sammy Robens-Paradise ID: 20709541

December 2, 2022

Systems Design Engineering, 2023
Department of Systems Design Engineering, University of Waterloo

Abstract – Website development has greatly improved since British computer scientist Tim Berners-Lee created the first graphical web browser in 1990 [2]. Today, software developers build a vast number of different applications made up of smaller applications all connected via the internet. Many of these allow humans to perform previously unattainable tasks right from their laptops. Of course, this change did not happen overnight. In the early days of the internet, people treated websites much as they did books, hence the term webpage. Therefore, a book with many pages would thus be a website with many webpages. When creating websites, the question then arises: "How would we design the website for a book with 300 pages, or even 3,000?". The answer is modularity, achieved by designing a system made of smaller well-documented components assembled together into a homogeneous application known as a *design system*. This report analyzes and seeks to make recommendations on the technical development of design systems based on decisions made by the front-end software engineering team at Beacon Biosignals (Beacon) in development of their GUI design system dubbed the *BECOs*.

Keywords – Design System, Documentation, Front End Infrastructure

I. SITUATION OF CONCERN & PROJECT OBJECTIVES

The term design system is often used loosely in the field of software engineering and recently has begun to adopt the ethos of a "buzzword". In practice however, a design system is far from "loose". It is "a collection of reusable components, guided by clear standards, that can be assembled together to build any number of applications" [4]. Within the context of this report discussing front end design systems (referring to the software development of user interfaces), a design system consists of user interface design constraints, software development constraints, and documentation constraints. Another critical distinction of design systems are their users. Whereas the primary users of graphical applications, such as a website, are near universally considered to be the end-users, the primary users of design systems are not the end-users but rather the developers, engineers and designers who build and maintain the application. This is because the base layer of a front end design system does not concern themselves with the appearance of a user interface, but rather the infrastructure that guides and constrains such an interface. End-users are considered secondary users in relation to design systems, and developers or designers working indirectly on the design system, such as via a dependency of the design system are considered tertiary users.

Beacon Biosignals was interested in exploring the usage of design systems to address the following needs identified by the core engineering team as a hindrance in the iterative product development process. 1) Large amounts code duplication. This was prevalent throughout the web client. For example, each button used in the web client had a unique signature and implementation. 2) Beacon Biosignals was in the process of growth and was under time constraint regarding the availability of designers to provide input, or design mock-ups for each unique design change. This resulted in often inconsistent GUI implementations, degrading the experience of secondary users (end-users). 3) Developers were spending valuable time updating existing code, which had to be done on a case-by-case bases since each signature was unique in description and implementation. 4), Developers wanted to ensure smaller components would behave as desired and required a safe environment to test them. Finally, 5), Beacon wanted to provide structure to the development process

so new designers and developers could quickly begin contributing with a low barrier of entry and could focus on UX-driven product development without the need to focus on top-down interface design.

The assumption was made that a design system would provide a framework to address these core problems that the engineering team at Beacon was facing. This was guided by the fact that Beacon's use case aligned closely with the principle use case described in Chapter 1 of Brad Frost's book, *Atomic Design* [6]. I was responsible for researching and developing a design system that would: reduce code duplication, enforce semantic design guidelines promoting web accessibility best practices, increase developer satisfaction and ensure quantitative code test coverage. Many of these metrics were subject to extreme variability caused by other software development efforts or would have required measurement over a long period. Without an extended observation period, making a distinction between the implementation of a design system and the holistic metric such as code duplication, would have been correlational at best. As a result, two metrics were chosen and narrowed down. The first was the qualitative level of developer satisfaction experienced when using the design system (The BECOs) derived from *Normans Principles of Good Design* [12]. This information was retrieved via one-on-one virtual meetings. The second was the quantitative code path test coverage enforced by the BECOs. This information was collected by *codecov*, a third-party industry standard integration that measures the number of code paths triggered by a series of software tests [1]. A *codecov* percentage score greater then the current *codecov* percentage score of the web client would be considered a success. A number of assumptions were made when developing these metrics. Firstly, it was assumed that developer behavior remained constant throughout the development of the BECOs. Secondly, it was assumed that the level of satisfaction experienced by developers remained constant through the development process. For example, if process A resulted in satisfaction level B prior to the development of the BECOs, such would remain true post-development.

II DESIGN METHODS

Because the design system was to be integrated into the existing web-client, a number of engineering constraints were placed on the project to enable integration. 1) The design system had to use React.js of version 16 or greater as its JavaScript framework. 2) The design system had to employ Typescript as a static compiler to ensure JavaScript (which is loosely typed) was statically typed [16]. 3) the design system source code needed to be trans-piled as part of the build process to ensure JavaScript is executable in a browser environment [18]. 4) The design system had to use tailwind-CSS, a CSS framework designed to improve CSS development that was used by the exiting web-client.

Due to the often fast-paced nature of software development at Beacon Biosignals, there was a need for engineering design methods to quickly highlight results without the need for extensive investigation and analysis. In order to achieve this, I, alongside the engineering team employed the usage of a computational decision matrix and rapid prototyping (for more complex decisions) to determine the most viable project architecture that would meet our engineering constrains and the needs of primary users. Given the initial engineering constraints, there were three core decisions that made using these design methods: the JavaScript bundler, which was chosen via a computational decision matrix, the package manager and repository structure, which were chosen via rapid prototyping.

Sonja Laurila concisely summarizes JavaScript bundlers in her thesis by stating "bundlers process application code, creating a dependency graph based on imports and exports recursively, which will then contain all the modules needed by the application. It will then bundle them into one or multiple bundle files, then injected into the HTML file [...] and run by the browser" [8]. Modularization is a key aspect of design systems and allows the separation of code into smaller more consumable and re-usable chunks. Most browsers, however, have no concept of code *modularization*. Bundlers allow us to describe modularized code in a way a browser can understand, critical to design systems.

To determine the appropriate package manager for the problem space, the two most popular bundlers were ranked on a 5-point scale of range [-2,2] where 0 indicated that the impact was negligible, "+" indicated a positive relationship and "-" indicated a negative relationship. The two options, rollup.js and webpack.js

were ranked against weighted categories derived from the needs-assessment. Each metric was given a weight range [1,3] where 3 was of the most importance, and 1 the least. The performance of each bundler was determined by analyzing benchmark data for each bundler [13] (Table 1).

Table 1: Webpack versus Rollup.js Computational Decision Matrix [Source: SRP, 2021]

Metric	Source code optimization (Tree shaking).	Documents and community support.	Backwards compatible support for ECMAScript Modules.	Low barrier of entry. Little experience with bundlers required to use.	Ability to customize	Results
Weight	1	3	2	3	2	-
webpack	+2	+2	-1	-1	+2	+7
rollup.js	+2	+2	+1	+1	-1	**+11**

The weighting was multiplied by the relationship and summed for each indicator to determine the best approach. The most desirable approach was the one with the highest score defined by:

$$result = \sum weight \cdot relationship$$

Rollup.js achieved a greater result, indicating that it was the preferred candidate largely because it had a positive relationship in regard to the barrier of entry. It features a simple API that relies on plugins to abstract away complexities, making Rollup.js easier to use than webpack [15]. Ease of use was deemed of high importance regarding development of the design system. After the establishment of a bundler, two low fidelity, two medium fidelity, and one high fidelity prototype were constructed as part of the design methodology to test the remaining two key infrastructure decisions. The first was the repository model that the design system (BECOs) should adhere to, and the second was the node.js package manager that the system should use to manage its dependencies.

There were two repository models that were under investigation as viable solutions for the BECOs. The first was a monorepo, meaning the BECOs, alongside other Beacon Biosignals applications would exist in the same central repository known as the platform. Beacon had traditionally taken a monorepo approach to software project management as it provides a single source of truth and encourages company-wide code collaboration [7]. The following is the (simplified) low fidelity prototype outlining a monorepo approach (Figure 1).

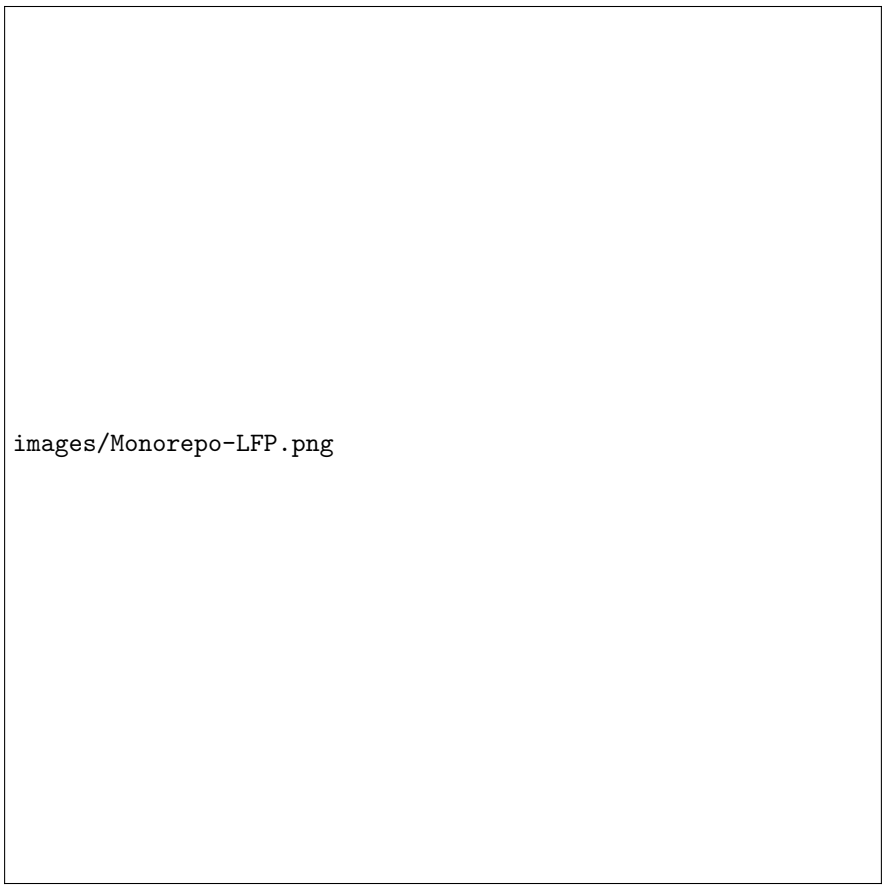

Figure 1: BECOs monorepo low fidelity prototype model [Image source: SRP, 2021]

Based on the low fidelity prototype, there was sufficient evidence to suggest that adhering to the monorepo approach showed promise, so a decision was made to develop, and open-source a medium fidelity prototype available on GitHub: SammyRobensParadise/spicy-mono, which contains instructions on how to re-construct the prototype from scratch. The prototype's reference name was dubbed *Spicy-Mono*

The second prototype used a multi-repo approach, where each project (in this case the BECOs) would inhabit its own repository. The advantages of a multi-repo approach are that it provides distinct separation between projects, and ownership. The follow is the low fidelity prototype outlining the multi-repo approach (Figure 2).

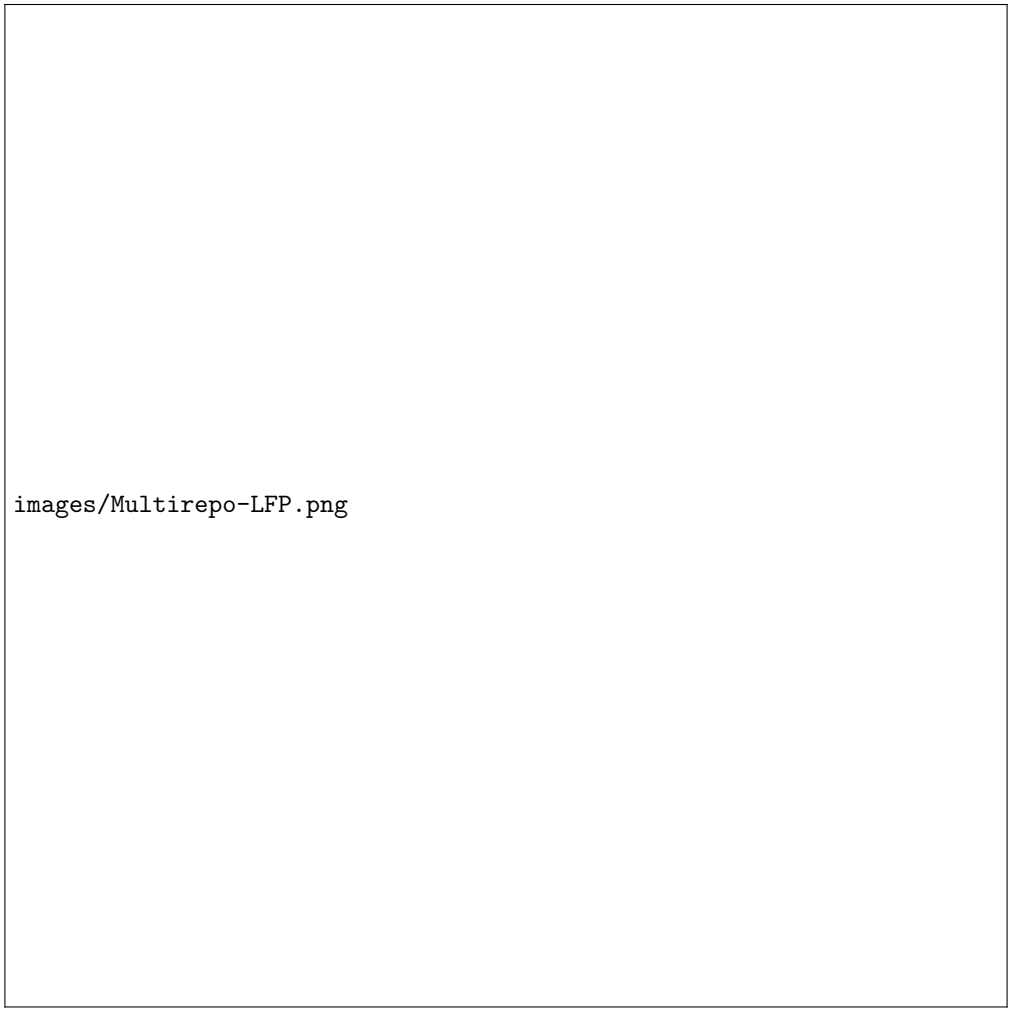

Figure 2: BECOs multi-repo low fidelity prototype model [Image source: SRP, 2021]

A prototype-based approach was also taken to determine the ideal package manager that could support the BECOs. There were two package managers that were considered. NPM (Node Package Manager) and Yarn, which was then subdivided by version since Yarn 2 (Yarn Berry) deviates heavily from Yarn 1 (Yarn Classic). These package managers were selected since they were the most popular node.js package managers at the time [3]. There are a number of key differences between these three package managers, most notably in the way Yarn 2 conducts dependency management versus Yarn 1 and NPM. Yarn 2 provides updated support for Yarn Workspaces, which is designed to allow multiple projects to exist as workspaces within the same repository, supporting the monorepo model [19]. Yarn 2 also generates a single file instead of a node_modules directory containing copied dependency code. The file contains "various maps: one linking package names and versions to their location on the disk, and another one linking package names and versions to their list of dependencies" [20]. This means that the time required to install dependencies reduced from minutes to seconds because I/O file copies are not required for each dependency [20]. The downside to this model is that Yarn 2 needs to know of all dependencies to resolve them, which many not be possible, since many legacy packages used in the JavaScript ecosystem do not explicitly declare their peer dependencies.

Two medium fidelity prototypes were created. The first was developed on top of the previously prototyped *Spicy-Mono* (SammyRobensParadise/spicy-mono) and evaluated the performance of NPM version 6 and Yarn 1. The second prototype was called *Spicy-Library* and evaluated the performance of Yarn 2. It was developed and open sourced on GitHub: SammyRobensParadise/spicy-library. The prototypes were evaluated qualitatively by the engineering team based on the following criteria. 1) Ability to instantiate a

new project using the package manager, 2) The package manager's ability to support required dependencies, and 3) The overall time taken to develop the prototypes using the respective package managers.

III DESIGNED SOLUTION

The development of the BECOs using the prototype-based design methodology coupled with a computational decision matrix generated a number of key results. Firstly, rollup.js was chosen as the desired bundler for the design system. This is largely because rollup.js presents a simpler API, which lowers the barrier of entry for new developers, a core project objective of the BECOs design system. This was reflected in the cumulative score achieved by Rolllup.js (11) over Webpack (7) via the computational decision matrix.

The prototypes revealed some additional interesting discoveries regarding repository infrastructure, and package managers. Beginning with the repository infrastructure, the question was asked whether the BECOs should adhere to Beacon Biosignals practice promoting a monorepo approach or opt for a multi-repo approach. The *Spicy-Mono* prototype proved that a design system could in fact exist within the same repository alongside its dependencies. The prototype consisted of two core directories, the */app* which contained a bootstrapped application used for testing, and */library* which contained a template for the BECOs. The dependencies were "hoisted" out of each directory and were managed by *Lerna* in the root of the repository. *Lerna* is an open-source tool for managing JavaScript projects containing multiple packages [9]. However, problems began to arise when an investigation into continuous integration and continuous deployment began. It quickly became clear that large amounts of work would have been necessary to update existing software infrastructure to allow the BECOs' continuous integration to be independent of the web client. It would also make it impossible to make the BECOs open source at a later date, which was a long-term goal of the design system since it would be exist alongside private source code. A multi-repo approach vastly simplified the development of a continuous integration and continuous deployment pipeline since it had no dependencies, and also ensured that the location of the BECOs could be easily changed and could exist in isolation. For this reason, the team opted for the multi-repo approach.

In regard to package managers, the prototype *Spicy-Library* revealed the pros and cons of Yarn 2. The advantages of Yarn 2 became immediately apparent. The package manager was able to install dependencies within seconds and the project itself was significantly smaller on the order of several megabytes. However, the development process for the prototype slowed to a crawl when it became time to setup the test applications. This was because the JavaScript ecosystem is extremely interdependent, and some packages with dwindling maintainers do not list their dependencies. Yet, Yarn 2 needs to know this information in order to create a dependency tree, whereas traditional package managers do not need to know this information (Figure 3). Instead, they rely on node.js's require algorithm [11, 20].

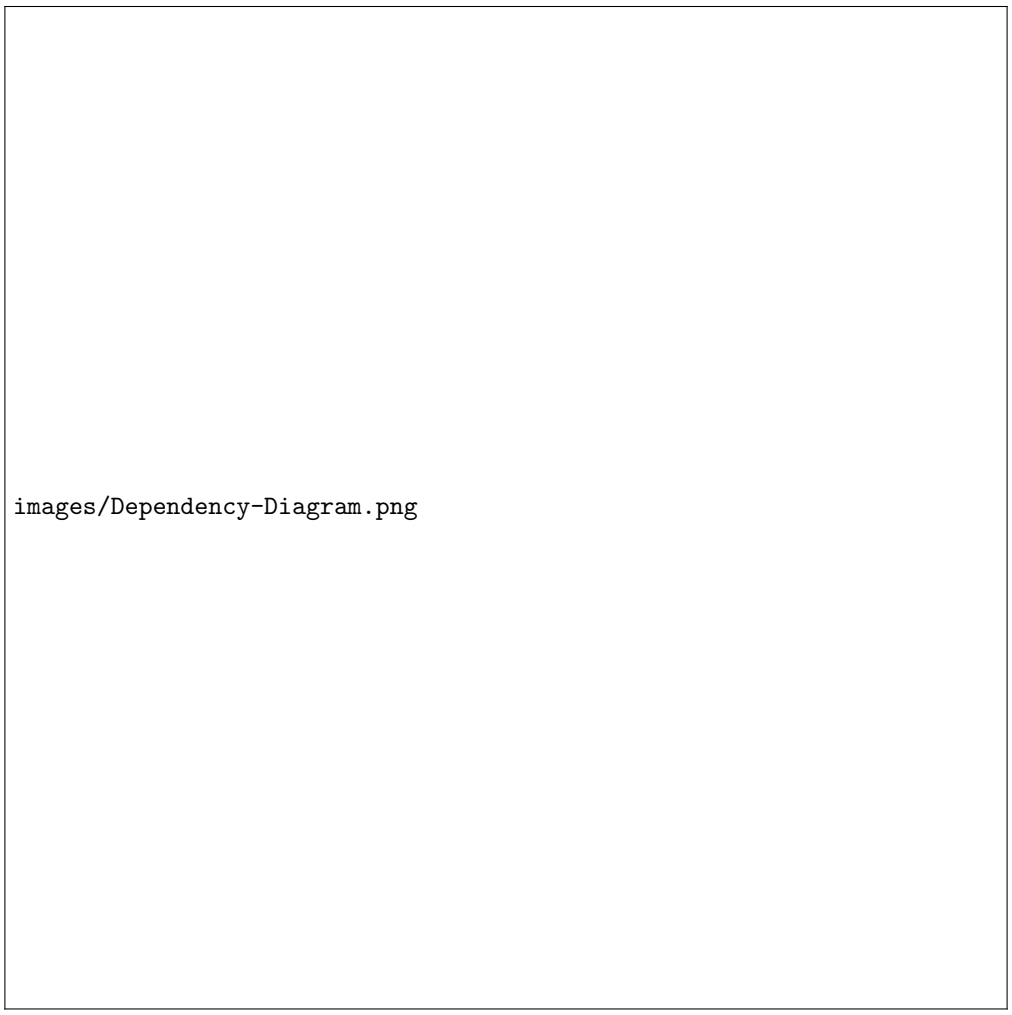

Figure 3: Dependency Diagram Required for Yarn 2 [Image source: SRP, 2021]

This presented a problem for the development of the BECOs, which relied heavily on a number of external packages which in turn had unlisted dependencies. The conclusion regarding Yarn 2 was that while it presented a unique and potentially powerful new direction for node.js dependency management, it did not yet have enough adoption for the BECOs' use case. This left Yarn Classic and NPM. Yarn Classic is a layer of abstraction over top of NPM designed to overcome some of the initial short-comings of NPM, however the latest versions of NPM are neck-in-neck with Yarn in terms of benchmarks. For this reason, NPM 6 was chosen as the package manager for the BECOs to reduce a redundant layer of abstraction from the development process.

There were a number of other technical decisions made as part of the development of the BECOs to align it with the requirements established by Beacon's engineering team at the beginning of the project. Notably, the introduction of Storybook, an "open-source tool for developing UI components and pages in isolation" [17]. Storybook provided an interactive interface for developers and designers to explore, build and test components before they are used in downstream applications, this was done to meet the design requirement ensuring developers were confident in the way components appeared and functioned before using them. Cypress and Jest, accompanied by React-Testing-Framework were also implemented and quantified using *codecov*. These tools allowed unit, and end-to-end tests to be executed on components before they were used as dependencies. This aligned with the requirement of the BECOs to have a *codecov* score greater than that of the web client at the time of the first BECOs release.

A vast number of additional packages and tools were used to ensure the BECOs met the engineering re-

quirements outlined as well as the needs of its intended primary users many of which were required my existing infrastructure.

Following these key decisions, the BECOs high fidelity prototype was instantiated in a separate repository using NPM 6 as it's package manager, the initial release of the BECOs contained the following UI components: avatars, buttons, check-boxes and checkbox groups, cloaks, over 200 icons, loaders, modals, notifications, protected text inputs, radio groups, spinners (spinning loading icons), text inputs, takeovers (UI overlays) and toggles (Figure 4). Continuous integration and automated deployments were developed alongside the operations team to ensure new code was thoroughly tested and tooling was implemented leveraging computer vision to detect code that may have caused visual regressions in UI components. It also contained a command line interface (CLI) tool designed to ease the experience of creating and scaffolding new components by auto generating a series of template files with one CLI command.

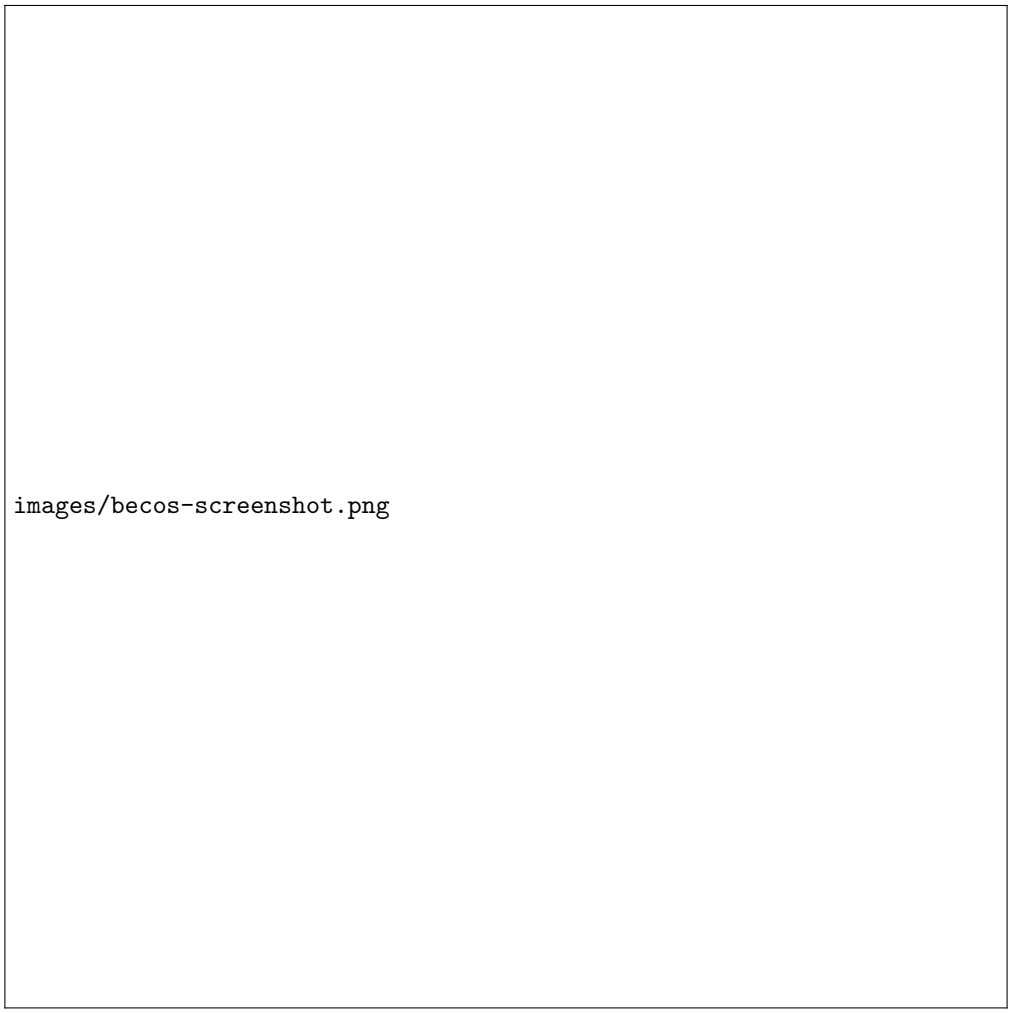

Figure 4: Screen Capture Taken from the BECOs Storybook Initial Release Featuring Toggle Components [Image source: SRP, 2021]

IV. DESIGN VALIDATION

Two key metrics were used to determine whether the BECOs had met it's intended objectives. The first what a qualitative analysis conducted to determine whether the BECOs had met the needs of developers who were to use the BECOs, in other terms, the primary users. The questions they were asked where derived

from some of *Norman's Principles of Good Design* [12]. The questions were chosen to provide insight into how well developers felt that the BECOs met their needs, as well as their mental model of a design system. Primary users were instructed to deliver strictly yes or no answers to each question to reduce ambiguity. 3 Developers participated. If they expressed confusion about a question an attempt was made to clarify the question. If confusion persisted, the question was skipped (Table 2).

Table 2: BECOs Developer Responses Based On Norman's Principles of Good Design [Source: SRP, 2021]

Norman's Principle	Discover-ability	Conceptual Model	Signifiers	Mappings (mental model)	Constraints
Associated Question	Can you determine what is achievable using the BECOs?	Documentation suggests how a product/sys-tem works.	Are the use cases outlining interactions with the BECOs clear?	Is the BECOs what you picture a design system to be?	Do you feel you have a good understanding of the limitations of the BECOs
Primary User A	Yes	Yes	Yes	Yes	Yes
Primary User B	Yes	Yes	Yes	Yes	Yes
Primary User C	Yes	Yes	Yes	Yes	Yes

Based on the answers provided by the three primary users of the BECOs, it became clear that the BECOs met all goals and was a design system that mapped well to the mental model of primary users, was discoverable and intuitive.

The second key metric was code coverage which was calculated by *codecov* using the following algorithm [1]:

$$CoveragePercentage = \left[\frac{\sum hits}{\sum hits + \sum partials + \sum misses}\right] \cdot 100\%$$

Where: Hit indicates that the source code was executed by the test suite. Partial indicates that the source code was not fully executed by the test suite [meaning] there are remaining branches that were not executed. Miss indicates that the source code was not executed by the test suite" [1].

The BECOs would meet this quantitative metric if and only if the code coverage of the BECOs was greater than that of the existing web client. This was to ensure that the introduction of design system to the existing web client would at least increase the code coverage by a percentage greater than 0. The code coverage percentage (score) of the web client at the time of the initial release of the BECOs was 55.60%. The code coverage percentage of the BECOs at this same instance was 99%. Based on these metrics, it is clear that integration of the BECOs into the existing web client would cause an increase in the code coverage of the web client by a value greater than 0, thus indicating that the engineering requirement was met given that:

$$A > B \Rightarrow n \cdot B < j \cdot B + k \cdot A, 0 < A, B, n, j, k < 1$$

Where: A represents the code coverage ratio of the BECOs, B represents the code coverage ratio of the web client, and n,j,k are arbitrary weights applied to each code coverage metric.

V. LIMITATIONS OF METHODS USED AND DESIGNED SOLUTION

The BECOs was subject to limitations in both design and analysis. Beginning with design, limitations came in the form of technological constraints. The design system was to be integrated with existing infrastructure meaning that it had to use similar technologies as the existing infrastructure, some of which was already considered legacy. Because of these inherit limitations, design solutions that ventured outside of these constrains were not researched or investigated. It is not known if or how an investigation into other tooling could benefit the technical implementation of the design system.

Analysis of the BECOs was constrained largely by time and the ability to gather a diverse set of primary users. Analysis of the BECOs took place shortly after its development, before it had been heavily integrated into existing infrastructure. This meant that certain key metrics such as, the number of developer-hours spend integrating the system, or the number of lines of code reduced by the introduction of the BECOs could not be determined. These metrics would help define the long-term value gained from the technical implementation of the BECOs. The analysis would have equally benefited from a larger group of primary users. This was because the three users available to answer questions concerning the BECOs were previously aware of or had actively contributed to the project. This introduced the potential for bias in their responses since they could have gained an understanding for the BECOs during its development rather than via usage of its high-fidelity prototype. Following the same course of logic, It is also not known whether user-participation in the development of the design system increased the likelihood that the design system would meet the needs of its core users, since a study with conditions close enough to those present during the development of the BECOs was not found, although James D. McKeen and Tor Guimaraes conclude in their 1997 paper, *Successful Strategies for User Participation in Systems Development* that user participation in the design process does (generally) have a positive relationship with user satisfaction [10].

VI. CONCLUSIONS

The initial situation of concern expressed that the BECOs must reduce code duplication, enforce semantic design guidelines promoting web accessibility best practices, increase developer satisfaction and ensure quantitative code test coverage. As previously stated, these requirements were to be evaluated based on the quantitative code coverage added by the BECOs, as the quantitative feedback collected from the primary users of the BECOs. The BECOs achieved a code coverage percentage of 99%, far surpassing the minimum code coverage requirement of 55.60%. All developers interviewed reported positively on their understanding of the BECOs, both in its limitations, usage and development. Based on these results, the BECOs fully met the needs outlined in the situation of concern, established when the project was initiated. This is largely due to the fact that technical decisions were made with the central goal of promoting ease of use. This influenced the decision to pursue rollup.js or webpack as the bundler, a multi-repo approach over a monorepo approach, and the usage of NPM 6 instead of Yarn (1 and 2) to reduce abstraction.

A larger conclusion can be inferred based on the implementation of the BECOs design system. The decision during the technical design portion of the project implementation to pursue a stack focused on high levels of support and simple APIs allowed for the BECOs to be easily understood by primary users and demonstrated alignment on key benchmarks with *Norman's Principles of Good Design* by promoting ease of use [12].

VII. RECOMMENDATION

Recommendation: Front end engineering teams should consider ease-of-use to be paramount when making technical decisions regarding the implementation of design systems. This is especially true at start-ups which can experience quick growth causing strain on intrinsic technical knowledge.
Rationale: Based on feedback collected from developers post-interaction with the BECOs, the focus on

technical documentation and ease of use was shown to be critical as 3 out of 3 developers indicated they had a complete understanding of the BECOs in response to questions derived from Norman's Principles of Good Design.

Costs: An additional focus on usability in regard to technical tooling added time to the holistic development process of the design system. In total 40 hours of developer-time were required to fully investigate all feasible design solutions via rapid prototyping. Given the average salary of a front end developer in Canada this would equate to an estimated $2,000.00 CAD [5].

Benefits: $2,000 CAD is a comparatively low-cost relative to the time that can be accumulated while attempting to fix problems caused by lack of technical documentation. It may also reduce the time required to orient primary users to company practices including software development and design, on which the average company spends over $4,000 CAD. [14].

ACKNOWLEDGMENTS

Thank you to Mike Buttery, Sr. front end Engineer at Beacon Biosignals, Seth Chapman, Software Engineer at Beacon Biosignals, Jarrett Revels, CTO and Co-founder at Beacon Biosignals, Chris De Graaf, Operations Engineer at Beacon Biosignals, and the extended team at Beacon for participating in countless project meetings, and the many hours dedicated to the development and creation of the BECOs.

References

[1] "About Code Coverage," *Codecov*, Dec-2020. [Online]. Available: https://docs.codecov.io/docs. [Accessed: 26-Apr-2021].

[2] "Browser History: Epic power struggles that brought us modern browsers," *Mozilla*. [Online]. Available: https://www.mozilla.org/en-US/firefox/browsers/browser-history/. [Accessed: 26-Apr-2021].

[3] M. Doyon, "JavaScript package managers compared: Yarn, npm, or pnpm?," *LogRocket Blog*, 23-Nov-2020. [Online]. Available: https://blog.logrocket.com/javascript-package-managers-compared/. [Accessed: 26-Apr-2021].

[4] W. Fanguy, "A comprehensive guide to design systems: Inside Design Blog," *A comprehensive guide to design systems*, 24-Jun-2019. [Online]. Available: https://www.invisionapp.com/inside-design/guide-to-design-systems/. [Accessed: 26-Apr-2021].

[5] "Front end developer Salary in Canada - Average Salary," *Talent.com*. [Online]. Available: https://ca.talent.com/salary?job=front end developer. [Accessed: 26-Apr-2021].

[6] B. Frost, *Atomic design*. Massachusetts: Brad Frost, 2016.

[7] C. Gehman, "What Is a Monorepo?," *Perforce Software*, 05-Mar-2020. [Online]. Available: https://www.perforce.com/blog/vcs/what-monorepo. [Accessed: 26-Apr-2021].

[8] S. Laurila, "Comparison of JavaScript Bundlers," *www.theseus.fi*, 10-Jun-2020. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/345959/Laurila_Sonja.pdf?sequence=2&isAllowed=y. [Accessed: 26-Apr-2021].

[9] Lerna, "lerna/lerna," *GitHub*, 10-Feb-2021. [Online]. Available: https://github.com/lerna/lerna. [Accessed: 26-Apr-2021].

[10] J. D. Mckeen and T. Guimaraes, "Successful Strategies for User Participation in Systems Development," *Journal of Management Information Systems*, vol. 14, no. 2, pp. 133–150, 1997.

[11] "Node.js v16.0.0 documentation," *Modules: CommonJS modules — Node.js v16.0.0 Documentation*. [Online]. Available: https://nodejs.org/api/modules.html. [Accessed: 26-Apr-2021].

[12] D. A. Norman, *The design of everyday things*. New York, NY: Basic Books, 2013.

[13] "Overview," *tooling.report*. [Online]. Available: https://bundlers.tooling.report/. [Accessed: 26-Apr-2021].

[14] A. Peterson, *The Hidden Costs of Onboarding a New Employee*, 26-Feb-2020. [Online]. Available: https://www.glassdoor.com/employers/blog/hidden-costs-employee-onboarding-reduce/. [Accessed: 26-Apr-2021].

[15] "Rollup," *rollupjs.org*. [Online]. Available: https://rollupjs.org/guide/en/. [Accessed: 26-Apr-2021].

[16] "Typed JavaScript at Any Scale.," *TypeScript*. [Online]. Available: https://www.typescriptlang.org/. [Accessed: 26-Apr-2021].

[17] "UI component explorer for frontend developers," *Storybook*. [Online]. Available: https://storybook.js.org/. [Accessed: 26-Apr-2021].

[18] "What is Babel? · Babel," *Babel*. [Online]. Available: https://babeljs.io/docs/en/. [Accessed: 26-Apr-2021].

[19] Yarnpkg, "Workspaces," *Yarn*. [Online]. Available: https://yarnpkg.com/features/workspaces. [Accessed: 26-Apr-2021].

[20] Yarnpkg, "Plug'n'Play," *Yarn*. [Online]. Available: https://yarnpkg.com/features/pnp. [Accessed: 26-Apr-2021].

Images, tables and graphs presented in this document created by the authors are indicated by author initials and the year of creation.