# Leaving the Tech Debt Behind: How to Sustainably Improve the User and Developer Experience of a Legacy Frontend by Designing, Building and Migrating to a New Web Client

Friedrich Maiwald[1][a], Nina Weber[1], Kay Massow[2][b] and Ilja Radusch[2]
[1]*Daimler Center for Automotive IT Innovations (DCAITI), Technische Universität Berlin, Berlin, Germany*
[2]*Fraunhofer Institute for Open Communication Systems (FOKUS), Berlin, Germany*

Keywords: Legacy Systems, Technical Debt, Software Maintenance, Software Modernization, Migration Strategy, Software Engineering, User Experience, Developer Experience, Web Development.

Abstract: Legacy web applications often suffer from declining user and developer experience, driven by technical debt, outdated technologies, and architectural complexity. This paper presents a structured approach for modernizing such systems, targeting simultaneous improvements in user experience (UX) and developer experience (DX). The process encompasses strategy selection, requirements elicitation and prioritization, technology evaluation, and construction best practices, all grounded in research and practical guidelines. Evaluation methods are defined for both UX and DX, combining established frameworks and actionable technical metrics. The approach is applied to the real-world migration of a legacy web application to a new frontend, using a gradual Strangler Fig strategy. The case study demonstrates how well-founded decisions with stakeholder involvement, modular architecture, modern tooling, and resilient testing can break free from legacy constraints. Quantitative and qualitative results show substantial gains in user satisfaction, codebase health, and developer productivity. The findings suggest that systematic modernization not only resolves immediate issues but enables sustainable, maintainable web applications. Future work should explore advanced quality assessment, long-term effects, and the integration of AI to support decision-making and automation in the modernization process.



## 1 INTRODUCTION

Legacy applications are older software systems that remain critical to the operations and success of the organizations that rely on them (Khadka et al., 2014). As these systems age, they tend to accrue substantial technical debt, which makes them increasingly difficult and costly to maintain (Abu Bakar et al., 2020). Technical debt includes the consequences of suboptimal design choices, outdated technologies, and accumulated complexities that hinder the system's performance and adaptability. This state of affairs has profound implications for both user experience (UX) and developer experience (DX) (Besker et al., 2020).

User experience refers to how efficiently, effectively, and satisfactorily users can complete their tasks using the system (Hassenzahl, 2008). A poor UX can lead to reduced productivity, frustration, and disengagement among users. Developer experience, on the other hand, pertains to the ease and motivation with which developers can maintain and extend the system (Fagerholm and Münch, 2012). A diminished DX can result in reduced morale, increased operational costs, and difficulties in responding to new requirements in a timely manner.

Given the critical role of legacy applications and the challenges posed by their maintenance, modernization strategies are essential. These strategies aim to address technical debt, improve system performance, and enhance both UX and DX. Modernization can involve various approaches, such as re-architecting the system, adopting new technologies, and refining the user interface.

This paper presents a comprehensive approach to modernizing legacy web applications, focusing on the implications of different modernization decisions. The approach is structured to include the crucial steps of an effective modernization process followed by suitable evaluation methods to make the outcomes verifiable as easily as possible. The goal is to provide a clear process and highlight the decision points

[a] https://orcid.org/0009-0009-3741-4307
[b] https://orcid.org/0000-0002-3760-5762



Maiwald, F., Weber, N., Massow, K. and Radusch, I.
Leaving the Tech Debt Behind: How to Sustainably Improve the User and Developer Experience of a Legacy Frontend by Designing, Building and Migrating to a New Web Client.
DOI: 10.5220/0013775500003985

In *Proceedings of the 21st International Conference on Web Information Systems and Technologies (WEBIST 2025)*, pages 213-219
ISBN: 978-989-758-772-6; ISSN: 2184-3252

that need to be considered, discussing the potential advantages and disadvantages of each choice.

To validate the proposed strategy, it is applied as a case study to the WebScE application, a web application for managing and visualizing mobility data. As a typical legacy system it is facing common challenges such as outdated technologies, accumulated technical debt, and complex codebases. The case study illustrates the modernization process, providing concrete examples of how to tackle various issues. Additionally, the effectiveness of the approach is evaluated by comparing the UX and DX of the current and modernized applications by applying the presented evaluation metrics.

By presenting this approach in a comprehensive and structured manner, this paper aims to offer valuable insights and practical guidance for similar modernization projects. While the case study is specific to the WebScE application, the principles and strategies discussed are expected to be applicable to a wide range of legacy web applications. The ultimate objective is to enhance the UX and DX of legacy systems, ensuring their continued relevance and effectiveness in meeting organizational needs.

The rest of this paper is organized as follows. In Section 2 we present the steps of our approach from specific modernization strategies, requirements analysis and technology decisions to architecture and construction best practices, all backed by intensive literature review. In Section 3, we describe the evaluation methods and specific metrics to profoundly assess the UX and DX of any modernization results. The approach and the evaluation methods are subsequently applied in the case study in Section 4. We finally conclude our findings in Section 5 and we give an outlook on the future work.

# 2 APPROACH

This section presents a structured, generalizable approach to modernizing legacy web applications, with the dual goal of improving user experience UX and DX. The approach balances the need for gradual renewal with maintainability and adaptability, and is divided into five key steps: modernization strategy, development process, requirements engineering, technology selection, and software construction best practices.

## 2.1 Modernization Strategy

Modernizing legacy systems typically involves choosing between a complete rewrite (”big bang”) and incremental refactoring of the existing system (Althani et al., 2016) (Stevenson and Pols, 2004). A full rewrite can eliminate technical debt but involves high risk and resource use, while incremental refactoring can be slow and demotivating for developers. Recent research and practical experience indicate that a middle ground, such as the Strangler Fig Pattern, provides a sustainable and less risky path (Ma et al., 2022).

In the Strangler Fig approach, new functionality is developed alongside the legacy application, with gradual migration of features to the new system (Fowler, 2004). This enables continuous user feedback, reduces business risk, and leverages modern technologies early. Integration strategies can include embedding parts of the legacy UI in the new frontend or providing seamless navigation to legacy modules (Verhaeghe et al., 2022). This ensures continuity and user acceptance during migration, while also allowing for early validation of UX and DX improvements.

A key insight from related work is that modernization should not simply replicate legacy features. Instead, it should leverage the opportunity to reassess requirements, address known pain points, and question the necessity of existing features. This helps avoid perpetuating legacy issues and supports both usability and long-term maintainability.

## 2.2 Development Process

Modernization projects have unique demands on the development process, as they must improve both UX and DX. A hybrid process model often combines the strengths of traditional, plan-driven methods (which support long-term code quality and maintainability — critical for DX) with those of agile methodologies (which foster continuous user feedback and iterative UX improvements) (Heimicke et al., 2020).

An effective approach begins with an initial plan-driven phase for high-level requirements, architecture, and technology decisions, providing a stable foundation. Subsequent development proceeds iteratively, prioritizing features based on value and cost-benefit considerations. Regular feedback from stakeholders and end-users is incorporated, enabling continuous refinement and early validation of UX and DX improvements. The process should remain flexible, adapting depth and documentation to the project's scale and team structure.

## 2.3 Requirements

Successful modernization requires not only reproducing existing functionality, but also explicitly identifying what must be improved (Kotonya and Som-

merville, 1998). The requirements phase should begin by eliciting the current pain points and desired improvements through techniques such as stakeholder interviews and explanatory observation. System archaeology — analyzing existing documentation and code — can supplement understanding, but should not be the sole source of requirements, as it risks perpetuating legacy flaws and unused features.

Requirements should be prioritized based on cost, benefit, and migration feasibility, often using analytical methods such as value/cost/risk matrices. Not all details should be fixed upfront; instead, high-level requirements are refined iteratively, in line with agile principles. Grouping related features into migration steps (as per the Strangler Fig pattern) helps plan incremental releases and ensures that each step delivers tangible value to users and developers alike.

### 2.4 Technology Selection

Selecting an appropriate technology stack is a critical decision in modernization, directly affecting both UX and DX. A structured technology selection framework should be applied (Kibanov et al., 2013) (Maxville et al., 2009), typically involving:

- **Criteria Specification:** Define clear, measurable selection criteria based on project goals, including usability, maintainability, community support, extensibility, testability, and documentation.
- **Mandatory/Desirable Criteria:** Distinguish between essential (must-have) and desirable (nice-to-have) features to efficiently narrow down candidates.
- **Weighting and Evaluation:** Assign weights to criteria according to their importance (e.g., maintainability and TypeScript support for DX, accessibility and theming for UX) and score candidates accordingly.
- **Final Selection:** Aggregate scores and make a final decision, considering both the quantitative results and qualitative fit with team experience.

Technologies should be chosen not only for their immediate fit, but also for their ecosystem maturity, long-term viability, and alignment with established community conventions. Frameworks and libraries that support resilient testing, enforce code conventions, and offer strong documentation can significantly ease onboarding and long-term maintenance.

### 2.5 Construction

The software construction phase emphasizes practices that ensure quality, maintainability, and adaptability:

- **Testing:** Automated tests are essential for sustainable modernization. Unit tests validate components in isolation, while end-to-end (E2E) tests verify complete user workflows and serve as living documentation for requirements. Tests should be resilient — robust against refactorings and changes in implementation details — to avoid maintenance overhead and encourage continuous improvement.
- **Code Conventions:** Enforcing consistent coding standards, ideally following widely-accepted community conventions (e.g., Clean Code (Martin, 2008)), enhances code readability and maintainability. Automated tools should be used to check and enforce these conventions.
- **Documentation:** Documentation should be concise, up-to-date, and maintained close to the codebase (e.g., within the project repository) (Sommerville, 2016). This ensures it remains relevant and easily accessible, reducing the risk of outdated information while facilitating onboarding and handovers.
- **Incremental Integration:** Where necessary, wrappers or abstraction layers around third-party libraries can reduce coupling and facilitate future replacements, but should be introduced judiciously to avoid unnecessary complexity.

By adhering to these principles throughout the construction phase, the modernization process not only delivers immediate UX improvements but also lays the foundation for long-term DX benefits, such as easier onboarding, maintenance, and future evolution.

## 3 EVALUATION METHODS

The success of a software modernization initiative must be evaluated against its primary objectives: enhancing user experience UX and DX. This section outlines a structured methodology — grounded in current research and best practices — for assessing improvements in both dimensions. The proposed approach combines standardized, multidimensional frameworks with practical, replicable metrics, enabling a comprehensive comparison of legacy and modernized systems.

### 3.1 User Experience

User experience is a multidimensional concept encompassing pragmatic and hedonic qualities, such as effectiveness, efficiency, satisfaction, and aesthetics

(ISO 9241-210:2019, 2019). To reliably assess the impact of modernization on UX, a combination of subjective and objective evaluation methods should be applied. A proven method is the HEART framework (Rodden et al., 2010), which structures UX assessment along key dimensions: Happiness (user satisfaction), Engagement, Adoption, Retention, and Task Success. For modernization projects, the most relevant dimensions are typically Happiness, Adoption, and Task Success.

- **Subjective Assessment:** Administer standardized UX questionnaires to capture users' perceived satisfaction, ease of use, and impression before and after modernization (Díaz-Oreiro et al., 2019).
- **Task-Based Usability Testing:** Conduct controlled usability experiments in which representative users perform typical tasks on both system versions. Collect metrics such as task completion time (efficiency), task success rate and error frequency (effectiveness), and error recovery time.
- **Performance Measurement:** Measure objective performance indicators like page load times, system responsiveness, and interaction delays (Elberkawi et al., 2016).
- **Adoption and Retention:** Track changes in the proportion of users who switch to the new system and their willingness to continue using it.

This multi-method approach provides both quantitative and qualitative data, enabling a thorough assessment of whether the modernized system achieves significant improvements in UX.

### 3.2 Developer Experience

Developer experience encompasses technical, cognitive, and affective aspects of working on a software system (Greiler et al., 2023). For modernization projects, the technical and process-related factors are particularly actionable and measurable.

- **Codebase Health:** Assess maintainability using static code analysis metrics such as cyclomatic complexity (McCabe, 1976). Lower complexity generally indicates more modular, understandable code.
- **Tooling and Environment:** Evaluate the ease of setting up the development environment, the availability and integration of modern development tools, and the automation of repetitive tasks.
- **Build and Release Efficiency:** Measure build times, deployment times, and the speed of development feedback cycles. Reduced build and release times correspond to better DX.
- **Test Suite Quality:** Quantify the robustness of automated tests using code coverage (statement coverage) and track the presence of meaningful unit and end-to-end tests. Maintain a balance between high coverage and maintainable test suites (Dodds, 2019).
- **Developer Feedback:** Supplement technical metrics with developer surveys or interviews to capture subjective perceptions of productivity, motivation, and friction points (Storey et al., 2021).

By combining automated technical measurements with qualitative feedback, this methodology offers a comprehensive and replicable framework for evaluating the success of modernization efforts in terms of developer experience.

## 4 CASE STUDY AND RESULTS

The following section applies the proposed modernization strategy to the WebScE web application and evaluates the results after the redevelopment.

### 4.1 Case Study: WebScE to VueScE

WebScE (*Web Scenario Editor*) is an internally used web application originally developed in the SimTD project (Stübing et al., 2010) for planning and evaluating field tests for connected driving. It is developed and maintained on a part-time basis by several senior and junior developers, its user interface is shown in Figure 1. The successive extension with project-specific features across multiple research projects eventually led to a significant accumulation of technical debt in the TEAM project (Bellotti et al., 2019): its frontend was implemented using frameworks that are now outdated (GWT and GXT), resulting in increasingly poor maintainability, degraded performance, and a lack of extensibility. The codebase grew complex and inconsistent, documentation became outdated, essential developer tooling was no longer available, and framework updates became difficult. As a result, both UX and DX suffered, with users facing long loading times and unexpected behavior, while developers struggled to introduce changes in a safe and efficient manner, and it became nearly impossible to find new developers to work with the outdated technologies.

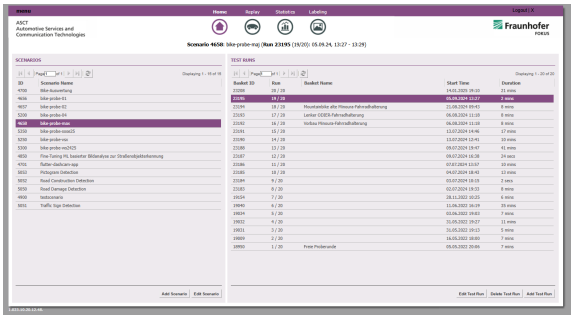

Figure 1: WebScE is the previous application, whose user interface looks outdated and is slow, and whose underlying code base is also outdated and difficult to maintain.

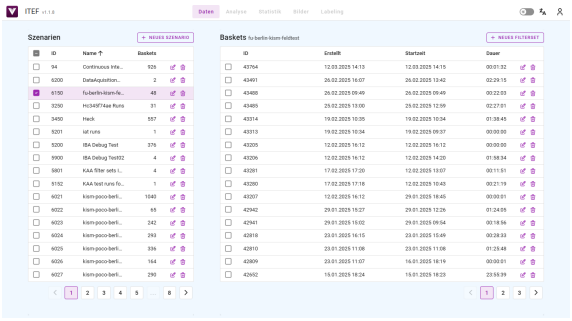

Figure 2: VueScE is the newly developed application. It adopts the previous WebScE interface to facilitate a quick transition for users, but is based on new, maintainable code.

## 4.2 Development Process Taken

To address these issues during the KIS'M project (Kwella et al., 2024) and add support for large-scale bicycle data (Massow et al., 2024), the modernization followed the approach described in Section 2. The chosen strategy was a gradual migration using the Strangler Fig pattern. Rather than refactoring the existing WebScE codebase, a new frontend application — VueScE as shown in Figure 2 — was developed from scratch using modern web technologies (TypeScript, Vue 3, and Ant Design).

The process began with an initial requirements elicitation phase, combining stakeholder interviews and explanatory observation to identify core user needs and pain points. Requirements were prioritized analytically, and only essential and high-priority features were included in the initial scope. Non-functional requirements focused on usability, efficiency, maintainability, and extensibility.

For technology selection, the proposed structured framework from Section 2.4 was applied, ensuring that the chosen stack would maximize both UX and DX. Vue 3 was selected as the application framework due to its modern architecture, moderate learning curve, strong community support, and excellent TypeScript integration. Ant Design was selected as the UI library for its rich component set and theming capabilities.

A modular architecture was adopted, grouping features into independent modules to facilitate maintainability and incremental evolution. The integration with the legacy WebScE was implemented via navigation: users could seamlessly switch to remaining legacy features directly from the new frontend, supporting early adoption and continuous feedback.

Development followed a hybrid process model (Water-Scrum-Fall (West, 2011)), combining plan-driven phases (for foundational decisions and architecture) with agile, iterative feature development and regular stakeholder reviews. Automated tests, code conventions and concise in-repository documentation were established from the beginning.

## 4.3 Results

The effectiveness of the modernization was evaluated using the methods described in Section 3, comparing the legacy WebScE and the new VueScE.

**User Experience:** The HEART framework was used to assess UX improvements. User satisfaction surveys showed a substantial increase, with the median satisfaction rising from 3.5 (WebScE) to 9 (VueScE) out of 10. Adoption and retention rates for the new frontend were high for available features. Task-based usability experiments demonstrated significant improvements: users were able to complete representative workflows up to 10 times faster in VueScE, primarily due to reduced loading times and a more intuitive interface. Objective performance measurements confirmed that page load times and navigation were greatly reduced.

**Developer Experience:** Developer experience was assessed via technical metrics and surveys. The codebase health improved dramatically: the percentage of functions with high cyclomatic complexity dropped below 1%, and the average complexity was significantly lower in VueScE. Modern developer tools and a simplified setup process increased productivity and reduced onboarding time. Build and release cycles became much faster — production builds were five times faster, and development builds enabled instant feedback. Automated test coverage increased to over 50% for unit tests, with meaningful E2E tests supporting safe refactoring. Developer surveys reflected much higher satisfaction (from 3 to 8 out of 10). Importantly, the anticipated long-term maintainability also began to manifest: two new developers were able to efficiently familiarize themselves with the code, successfully replace the originally selected

UI library Ant Design by Vuetify, and expand the modular structure with new features.

In summary, applying the described modernization approach to the migration from WebScE to VueScE resulted in substantial improvements to both user and developer experience, validating the effectiveness of the proposed process for legacy web application renewal.

# 5 CONCLUSION

This paper presented a structured approach for modernizing legacy web applications, with the dual objective of improving both UX and DX. The approach was grounded in current research and best practices, covering all stages from modernization strategy and requirements engineering to technology selection, construction, and evaluation. By applying this process to the migration from WebScE to VueScE, the case study demonstrated substantial improvements in both UX and DX, as evidenced by quantitative and qualitative evaluation metrics.

The results show that modernization — when approached systematically — can successfully overcome the typical challenges of legacy systems, such as technical debt, poor maintainability, and degraded user and developer satisfaction. The described modernization approach enabled not only short-term improvements in usability and productivity, but also laid the foundation for a sustainable, long-term maintainable software system. Key success factors included early and continuous stakeholder involvement, prioritization of requirements, use of modern and well-supported technologies, modular and extensible architecture, and the establishment of resilient testing and documentation practices.

Applying this approach to other similar modernization projects in addition to this case study would be useful for a more comprehensive assessment.

## 5.1 Future Work

While this paper demonstrates substantial improvements in UX and DX through systematic modernization, several open questions and promising research directions remain. First, the field of developer experience (DX) assessment is still in its infancy. Although initial frameworks for quantitative measurement exist, further validation is required to ensure reliability and generalizability across contexts. Developing new, robust, and practical DX metrics, as well as investigating their relationship to long-term productivity and team satisfaction, represents an important avenue for future work.

Second, quality assessment methods for code, test suites, and documentation remain limited in scope and tooling. Metrics such as cyclomatic complexity and code coverage are widely used but have known weaknesses. Future research should focus on creating more meaningful, language-agnostic metrics and affordable tooling for code quality, mutation testing, and especially documentation quality (Treude et al., 2020). Automated, objective measurement approaches would benefit both scientific evaluation and industry practice.

Third, the long-term effects of modernization approaches merit further longitudinal studies. While initial results are promising, repeated UX and DX assessments over time could provide valuable evidence about the sustainability and maintainability of the modernized solution, and the real impact of architectural and process decisions.

Fourth, the integration of artificial intelligence (AI) into modernization processes is a promising but largely unexplored field. AI could support decision-making in requirements analysis, technology selection, or automated quality analyses. However, the effective and responsible use of AI tools in modernization workflows needs systematic investigation in terms of technical capabilities and human oversight.

Finally, as modernization projects are often highly context-specific, further case studies in diverse domains and settings will help to generalize and refine the proposed approach. Collaboration between academia and industry can accelerate knowledge transfer and the development of best practices for sustainable software modernization.

# ACKNOWLEDGEMENTS

Relevant parts of the literature review, the developed modernization approach and the implementation of the VueScE application originate from the thesis (Weber, 2024), which was supervised by the authors of this paper. An AI tool was used to check the wording, grammar, and spelling in this paper.

# REFERENCES

Abu Bakar, H. K., Razali, R., and Jambari, D. I. (2020). A guidance to legacy systems modernization. *International Journal on Advanced Science, Engineering and Information Technology*, 10(3):1042–1050.

Althani, B., Khaddaj, S., and Makoond, B. (2016). A quality assured framework for cloud adaptation and modernization of enterprise applications. In *2016 IEEE*

*Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 634–637.

Bellotti, F., Kopetzki, S., Berta, R., Paranthaman, P. K., Dange, G. R., Lytrivis, P., Amditis, A. J., Raffero, M., Aittoniemi, E., Basso, R., Radusch, I., and De Gloria, A. (2019). Team applications for collaborative road mobility. *IEEE Transactions on Industrial Informatics*, 15(2):1105–1119.

Besker, T. et al. (2020). The influence of technical debt on software developer morale. *Journal of Systems and Software*, 167:110586.

Dodds, K. C. (2019). Write tests. not too many. mostly integration. https://kentcdodds.com/blog/write-tests. visited on Oct. 13, 2023.

Díaz-Oreiro, I. et al. (2019). Standardized questionnaires for user experience evaluation: A systematic literature review. In *Proceedings 31.1*, page 14.

Elberkawi, E. K. et al. (2016). Usability evaluation of web-based systems: A new method and results. In *2016 International Conference on Engineering & MIS (ICEMIS)*, pages 1–5.

Fagerholm, F. and Münch, J. (2012). Developer experience: Concept and definition. In *Proceedings of the International Conference on Software and System Process (ICSSP)*, pages 73–77.

Fowler, M. (2004). Strangler fig application. https://martinfowler.com/bliki/StranglerFigApplication.html. visited on June 30, 2025.

Greiler, M., Storey, M.-A., and Noda, A. (2023). An actionable framework for understanding and improving developer experience. *IEEE Transactions on Software Engineering*, 49(4):1411–1425.

Hassenzahl, M. (2008). User experience (ux): Towards an experiential perspective on product quality. In *Proceedings of the 20th Conference on l'Interaction Homme-Machine*, pages 11–15.

Heimicke, J., Chen, R., and Albers, A. (2020). Agile meets plan-driven – hybrid approaches in product development: A systematic literature review. In *Proceedings of the Design Society: DESIGN Conference*, volume 1, pages 577–586.

ISO 9241-210:2019 (2019). Ergonomics of human-system interaction — part 210: Human-centred design for interactive systems. Standard, International Organization for Standardization. https://www.iso.org/standard/52075.html.

Khadka, R. et al. (2014). How do professionals perceive legacy systems and software modernization? *Proceedings of the 36th International Conference on Software Engineering*, pages 36–47.

Kibanov, M., Erdmann, D. J., and Atzmüller, M. (2013). How to select a suitable tool for a software development project: Three case studies and the lessons learned. In *Software Engineering 2013 - Workshopband*, pages 415–424.

Kotonya, G. and Sommerville, I. (1998). *Requirements Engineering: Processes and Techniques*. John Wiley & Sons.

Kwella, B., Massow, K., Schaeufele, B., Radusch, I., Hark, J. N., Ritter, C. N., Cao, B., Kleinert, M., Wille, C., and Strop, O. (2024). Implementation and testing of v2x-applications for near future urban traffic in berlin. In *2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–5.

Ma, S.-P., Li, C.-Y., Lee, W.-T., and Lee, S.-J. (2022). Microservice migration using strangler fig pattern and domain-driven design". *Journal of Information Science and Engineering*, 38(6):1285–1303.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.

Massow, K., Maiwald, F., Thiele, M., Heimendahl, J., Protzmann, R., and Radusch, I. (2024). Crowd-based road surface assessment using smartphones on bicycles. In *2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)*, pages 1–8.

Maxville, V., Armarego, J., and Lam, C. (2009). Applying a reusable framework for software selection. *IET Software*, 3(5):369–380.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Rodden, K., Hutchinson, H., and Fu, X. (2010). Measuring the user experience on a large scale: User-centered metrics for web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*, pages 2395–2398.

Sommerville, I. (2016). *Software Engineering*. Pearson, 10th edition.

Stevenson, C. and Pols, A. (2004). An agile approach to a legacy system. In *Extreme Programming and Agile Processes in Software Engineering. XP 2004*, pages 123–129.

Storey, M.-A. et al. (2021). Towards a theory of software developer job satisfaction and perceived productivity. *IEEE Transactions on Software Engineering*, 47(10):2125–2142.

Stübing, H., Bechler, M., Heussner, D., May, T., Radusch, I., Rechner, H., and Vogel, P. (2010). simtd: a car-to-x system architecture for field operational tests [topics in automotive networking]. *IEEE Communications Magazine*, 48(5):148–154.

Treude, C., Middleton, J., and Atapattu, T. (2020). Beyond accuracy: assessing software documentation quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1509–1512, New York, NY, USA. Association for Computing Machinery.

Verhaeghe, B. et al. (2022). A hybrid architecture for the incremental migration of a web front-end:. In *Proceedings of the 17th International Conference on Software Technologies. ICSOFT*, pages 101–110.

Weber, N. (2024). Improving user and developer experience of a legacy web application. Master's thesis, Technische Universität Berlin, Berlin, Germany.

West, D. (2011). Water-scrum-fall is the reality of agile for most organizations today. https://www.forrester.com/blogs/11-07-26-water_scrum_fall_is_the_reality_of_agile_for_most_organizations_today/. visited on Feb. 26, 2024.