Critical Review Assignment

on

A Plea to Tool Vendors:
Do Not Mislead How Technical Debt Is Managed

By: Ipek Ozkaya
Editor in Chief at Carnegie Mellon Software Engineering Institute
ipek.ozkaya@computer.org

---

and

The Fog of Software Design

By: Timothy J. Halloran
Software engineer at Google, Pittsburgh, Pennsylvania, 15206, USA
Retired U.S. Air Force Lieutenant Colonel
hallorant@gmail.com

Reviewed by Group – 2

Team members details and contribution:

1. **Raman Sharma – 22114076**
   Contribution: Contributed to 'Key Contribution and Key Points from the Paper'
   Contact No: +91 75218 29455
   Email ID: raman_s@cs.iitr.ac.in

2. **Boda Yashwanth – 22114022**
   Contribution: Contributed to 'Techniques and Tools Used in Paper' and 'Solutions to Improve the Methods mentioned in the Paper'
   Contact No: +91 89779 25111
   Email ID: boda_y@cs.iitr.ac.in

3. **Ayush Ranjan – 22114018**
   Contribution: Contributed to 'Summary of the paper'
   Contact No: +91 74829 58551
   Email ID: ayush_r@cs.iitr.ac.in

4. **Souvik Karmakar – 22114096**
   Contribution: Contributed to ‘Advantages of Methods mentioned in Paper’
   Contact No: +91 86429 24659
   Email ID: souvik_k@cs.iitr.ac.in

5. **Sarvasva Gupta – 22114086**
   Contribution: Contributed to ‘Techniques and Tools Used in Paper’
   Contact No: +91 79899 04811
   Email ID: sarvasva_g@cs.iitr.ac.in

6. **Anvit Gupta – 22114009**
   Contribution: Contributed to ‘Issues and Loopholes in Paper’
   Contact No: +91 94620 11044
   Email ID: anvit_g@cs.iitr.ac.in

7. **Vineet Kumar – 22114107**
   Contribution: Contributed to ‘Issues and Loopholes in Paper’
   Contact No: +91 92634 36403
   Email ID: vineet_k@cs.iitr.ac.in

Summary

“A Plea to Tool Vendors: Do Not Mislead How Technical Debt Is Managed”

The article discusses the importance of managing technical debt in software engineering and calls for clarity in how it's addressed by tool vendors. It emphasizes that technical debt is about architecture and design trade-offs, distinct from defects and vulnerabilities. The author urges vendors not to conflate technical debt with other issues, to focus on architecture-related features, and to be transparent about their capabilities. Additionally, they advocate for including a specific "technical debt" issue type in bug tracking software. They argue that these shifts in tooling will lead to better identification and management of technical debt, ultimately improving software quality and development practices.

“The Fog of Software Design”

The article discusses the challenges in software design, likening them to a "fog" that obscures critical information. It highlights four main challenges: unclear requirements, futile attempts at predicting future needs, struggles with existing code, and limited design knowledge. First, requirements are often incomplete, leading to overlooked risks. Developers are advised to prioritize tackling risky design elements over comfortable work. Second, attempting to foresee future needs often leads to wasted effort. Infrastructure

projects should be approached cautiously to ensure necessity and reusability. Third, rewriting existing code often leads to misunderstandings of original design decisions. Gradual transition into production helps uncover vital insights and ensures project success. Fourth, inadequate design knowledge can hinder progress. Seeking feedback and embracing contentious design processes can lead to better outcomes by expanding knowledge and considering alternatives. In conclusion, the article recommends anticipating and preparing for uncertainty, utilizing design abstractions, iterating, and seeking feedback to navigate through the challenges effectively.

Key contribution and Key Points from the Paper

"A Plea to Tool Vendors: Do Not Mislead How Technical Debt Is Managed"

- **Technical Debt vs. Defects and Vulnerabilities:**
- Technical debt is distinct from defects (coding errors) and vulnerabilities (security weaknesses).
- While technical debt can increase defect proneness and vulnerability risks, it's not the same issue.
- Detecting code errors and security flaws shouldn't be marketed as managing technical debt.
- **Managing Technical Debt Effectively:**
- Tool vendors should avoid overgeneralizing architecture analysis and be clear about their capabilities.
- Issue and bug tracking tools should offer a dedicated "technical debt" category for clear identification and tracking.
- This shift will enable better data collection and research on quantifying and managing technical debt.
- **Call to Action for Tool Vendors:**
- Stop marketing features for detecting defects and vulnerabilities as "technical debt management."
- Clearly differentiate architecture analysis from security or performance analysis.
- Offer dedicated issue types for technical debt in issue tracking tools.
- **Impact:**
- Improved understanding and management of technical debt in software development.
- More accurate data and research on technical debt quantification and resource allocation.
- Empowering software teams to identify and express technical debt more effectively.
- **Additional Notes:**
- The paper acknowledges the potential benefits of tool support for detecting certain issues related to technical debt.

- However, it emphasizes the importance of accurate terminology and focusing on architecture and design aspects.
- Collaboration between researchers, industry practitioners, and tool vendors is crucial for effective technical debt management.

"The Fog of Software Design"

The article discusses the challenges software designers face due to limited information and uncertainties.

This is referred to as the "fog of software design" and includes:

- **Misunderstanding requirements:** Requirements are often unclear, incomplete, or evolve over time.
- **Difficulty anticipating future needs:** Predicting future requirements is challenging and often leads to wasted effort.
- **Limited understanding of existing code:** When rewriting or modernizing existing code, the original design decisions might be unclear or forgotten.
- **Lack of design knowledge:** Designers may not be familiar with all potential design approaches, limiting their options.

The author suggests several approaches to deal with the fog of software design:

- **Avoid "panic and hack":** Rushing into implementation without proper design leads to long-term problems.
- **Be aware of limitations of academic and theoretical approaches:** These perspectives often oversimplify real-world complexities.
- **"Fake" a rational design process:** This approach (by Parnas and Clements) involves creating detailed documentation to compensate for the messy reality, but the author finds it impractical for most projects.

The author's recommendations for navigating the fog include:

- **Using design abstractions and iterating:** Break down the problem and gradually refine the design through iterations.
- **Focusing on risks:** Identify and prioritize the riskiest parts of the design and address them first.
- **Tailoring documentation:** The level of formality and detail in documentation should be adapted to the specific project and domain.
- **Seeking feedback:** Actively solicit feedback on your design from others to gain different perspectives and improve the outcome.
- **Maintaining composure:** Don't be discouraged by the fog; anticipate it, be prepared, and keep working through the challenges.

Overall, the article emphasizes the importance of acknowledging the inherent uncertainties in software design and adopting practical strategies to navigate them effectively.

Tools and Techniques Used in Paper

The article mentions many modern software tools used to maintain software quality, analyse code and track bugs and issues in complex software efficiently. These are:

- **Issue and bug tracking software tools** like Jira, GitLab, Team Foundation Server, Bugzilla, etc. These are tools which help teams manage and resolve problems, bugs, and tasks identified during software development and operational activities.

- **Static code analysis tools** like SonarQube and Checkmarx SAST help to analyse computer software without actually running the software.

- **Software quality management tools** like Qualio and Unifize ensure software products meet required standards and fulfil user expectations. It encompasses techniques like quality planning, quality control, and quality assurance.

- **Java dynamic analysis tools** which help you analyse your code during runtime. These tools allow you to observe program behaviour, identify performance bottlenecks, memory leaks, and other issues.

- **'Fake rational design' technique** to deal with complex code: Instead of trying to untangle the existing mess, they suggest creating new, well-documented designs that pretend the code was built following a clean and organized process. This new documentation would explain the "fake" design and how the existing code fits into it, even though it might not be entirely accurate.

Advantages of Methods mentioned in Paper

"A Plea to Tool Vendors: Do Not Mislead How Technical Debt Is Managed"

GitLab:

- Tracks code changes and identifies problematic areas.
- Improves code quality through code reviews and static analysis.
- Automates testing to catch regressions early.

Jira:

- Tracks and prioritizes technical debt issues.
- Provides clear workflows and ownership for addressing them.
- Offers insights for data-driven decision making.

Combining GitLab and Jira offers several advantages for tackling technical debt:

- Improved visibility and tracking: GitLab for code history and Jira for dedicated issue management.
- Enhanced collaboration: GitLab for code reviews and Jira for communication between teams.
- Streamlined workflow: Integrations with tools for code analysis and automated testing in GitLab and linking code changes to Jira issues.

This combination can:

- Increase awareness: Raise awareness of technical debt within the team.
- Improve code quality: Track and manage technical debt effectively to prioritize improvements and deliver higher quality software.
- Enhance decision-making: Leverage data from both tools for informed decisions about resource allocation and technical debt fixes.

"The Fog of Software Design"

This article outlines a pragmatic and adaptable approach to tackling the challenges of designing software in the face of uncertainty:

1. Accepting and Anticipating Uncertainty:

- Acknowledge the inherent limitations of complete knowledge in a project.
- Anticipate encountering "foggy" areas with incomplete information or unclear requirements.

2. Embracing Iteration and Abstraction:

- Leverage design abstractions to decompose complex problems into manageable units.
- Employ an iterative design process that allows for continuous refinement as knowledge evolves.

3. Risk-Driven Prioritization:

- Prioritize addressing high-risk elements with the most potential to negatively impact the project.

- This proactive approach helps mitigate future problems and ensures efficient resource allocation.

4. Tailored Documentation:

- Avoid overly elaborate documentation that can become obsolete quickly.
- Instead, tailor the level of documentation formality to the specific needs and context of the project.

5. Fostering Collaboration and Feedback:

- Encourage active discussion and solicit feedback on design decisions to identify issues early on.
- Collaborative design fosters diverse perspectives and helps create more robust solutions.

While the "fog" of design is an ever-present challenge, this practical approach equips software developers with essential tools and strategies. By embracing iterative processes, prioritizing risk mitigation, and fostering collaboration, developers can navigate the uncertainties of design and deliver successful software solutions.

Issues and Loopholes in Paper

"A Plea to Tool Vendors: Do Not Mislead How Technical Debt Is Managed"

- **Oversimplification:** Static code analysis tools offer limited insight and miss the root cause of technical debt.
- **Misinterpretation Risk:** Explicitly categorizing technical debt in issue trackers can lead to under-reporting of critical bugs.
- **Limited Scope:** The paper overlooks development practices, prioritization, and team dynamics in managing technical debt.
- **Commercial Tool Bias:** The paper prioritizes expensive commercial tools, neglecting open-source alternatives and human expertise.
- **Small Team Issues:** The recommended approach might not be suitable for small teams with limited resources.
- **Limited Software Applicability:** Static code analysis tools may not be effective for all software types.
- **Downplaying Human Judgment:** The paper overemphasizes tools, potentially neglecting the importance of human judgment and best practices.

"The Fog of Software Design"

- **Limited Applicability:** Recommendations might not directly apply to greenfield projects.
- **Increased Complexity:** Breaking down systems can increase complexity in managing dependencies and integration.
- **Resource-intensive:** Rigorous design and extensive feedback might not be feasible for small teams or tight budgets.
- **Documentation Scalability:** Extensive formal documentation might not scale for larger projects and can become outdated.
- **Lack of Tool Integration:** Integrating the paper's suggestions with existing development tools could improve applicability.

Solutions to Improve the Methods mentioned in the Paper

"A Plea to Tool Vendors: Do Not Mislead How Technical Debt Is Managed"

Following are some solutions to address the issues and limitations identified in the paper:

- **Expand on the types of technical debt:** Go beyond basic definitions and showcase examples of common technical debt categories, such as code duplication, lack of documentation, or poor architecture choices.
- **Acknowledge the spectrum of severity:** Explain that not all technical debt is equally critical. Some may have minimal impact, while others can significantly hinder future development.
- **Focus on core principles:** Shift the emphasis from specific tools to the underlying principles of issue management. This allows users to adapt the principles to their existing tools and workflows.
- **Emphasize the need for context awareness:** Stress the importance of understanding the specific context and project goals when evaluating and prioritizing issues, something that solely automated tools might struggle with.
- **Suggest alternative approaches:** Explore alternative strategies or adaptations of existing methods for handling different software types, catering to a wider range of software projects.

"The Fog of Software Design"

Following are some solutions to address the issues and limitations identified in the paper:

- **Illustrate with diverse examples:** Showcase the "fogs" of software design in various contexts, not just one specific domain, to demonstrate broader relevance and applicability.
- **Advocate for concise and focused documentation:** Encourage the creation of concise and focused documentation that captures key design decisions and intent, rather than exhaustive documentation that becomes cumbersome to maintain.
- **Offer pragmatic integration suggestions:** Instead of proposing entirely new methods, explore how the ideas can integrate with existing design and documentation practices to minimize disruption and encourage adoption.
- **Propose lightweight solutions:** Suggest methods that don't require significant resource investment, like collaborative design workshops or knowledge-sharing platforms, to make the approach more accessible.

References

1. Ipek Ozkaya (2021). *A Plea to Tool Vendors: Do Not Mislead How Technical Debt Is Managed*. IEEE Computer Society.

2. Timothy J. Halloran (2021). *The Fog of Software Design*. IEEE Computer Society.

3. “Technical Debt”. ProductPlan. https://www.productplan.com/glossary/technical-debt/

4. “Technical Debt”. Wikipedia. https://en.wikipedia.org/wiki/Technical_debt