Universal Library of Engineering Technology Research Article

ISSN: 3064-996X | Volume 3, Issue 1

Open Access | PP: 60-64

DOI: https://doi.org/10.70315/uloap.ulete.2026.0301010

The Impact of Artificial Intelligence on Web UI Development

Yurii Bezhentsev

Software Developer, Houston, Texas, United States.

Abstract

The article examines how generative artificial intelligence is reshaping Web UI development, primarily by reconfiguring the infrastructure layer, interface component libraries, and design systems, thereby transforming not the speed of layout implementation but the economics of reusable artefacts. The aim of the study is to provide a conceptual explanation of why, under the mass adoption of LLM-based assistants and intelligent interface methods, legacy monetization models based on selling ready-made themes, templates, screen collections, and even paid access to components begin to lose stability. The relevance of the work stems from the fact that contemporary web development functions as a dependency network of package ecosystems, where component libraries act as channels for scaling practices, while AI drastically reduces the cost of obtaining a first working version and reproducing standard solutions, simultaneously increasing the variability of results and the risk of divergence between teams. The novelty of the article lies in shifting the focus from the technological effect of generation to the shift in the unit of value: it is shown that in conditions of commoditization of form, market power moves toward what cannot be generated in a single pass, toward the manageability of the design system, verifiable contracts (tokens, standards, rules of composition), design–code integrations, automated quality control, and guarantees of compatibility and predictable updates. The main conclusion is that the winners are not vendors of yet another button, but platforms that sell trust, disciplined change, and reproducible UI evolution. The article is intended for developers, design leads, product managers, and creators of design systems who are choosing strategies for sustainable monetization in the era of generative AI.

Keywords: *Artificial Intelligence, Generative Models, Web UI, Design Systems, Component Libraries, Monetization.*

INTRODUCTION

Interface component libraries and design systems have become the infrastructure of web development, not because they speed up markup, but because they constitute the reusable foundation of a product: standard elements, rules of composition, tokens, and agreements on accessibility are transformed into a stable layer on which teams and software supply chains rely. This infrastructural nature is clearly visible at the level of package ecosystems: modern development is built as a network of dependencies, in which even small libraries serve as transit nodes. For a package repository in a web development language, it has been shown to contain more than 3 million packages and to serve tens of billions of downloads per week. That is, the library layer becomes a mass channel for disseminating and standardizing practices (Pinckney et al., 2023).

Against this background, artificial intelligence functions as an accelerator that alters the economics of precisely infrastructural artefacts. Systematic reviews of language models for code autocompletion report that such models are already embedded in development environments and noticeably improve development effectiveness, reducing the cost of obtaining a first working version and lowering the barrier to reproducing standard solutions (Husein et al., 2025). In parallel, research on intelligent interfaces demonstrates a stable trend toward transferring part of design and evaluative activities into computational methods, meaning that automation affects not only programming but also the making of interface decisions (Brdnik et al., 2022). As a result, what used to be sold as scarce expertise, privileged access to components, or ready-made screen sets is increasingly perceived as an easily reproducible outcome that can be generated and refined.

From this follows the objective of this article: AI is changing the rules of the game by weakening the monetization of interface components that the economy has relied on until now: instead of packaged standard solutions, the economy must offer guarantees of quality, stability, and change. This finding aligns with the conclusion that open software components are a public good whose economic

Citation: Yurii Bezhentsev, "The Impact of Artificial Intelligence on Web UI Development", Universal Library of Engineering Technology, 2026; 3(1): 60-64. DOI: https://doi.org/10.70315/uloap.ulete.2026.0301010.

costs are difficult to assess and are burdened by the chronic sustainability problem (Carter, 2024).

MATERIALS AND METHODOLOGY

The study of the impact of artificial intelligence on Web UI development is based on an analysis of a corpus of eight key sources, including empirical and review works on LLM-based assistants in software development, research on intelligent user interfaces, and literature on the economics and business models of open source software and the infrastructure of package ecosystems. The theoretical framework is built along two intersecting lines: (1) the conceptualization of component libraries and design systems as an infrastructural layer that scales practices via dependency networks and mass package distribution (Pinckney et al., 2023), and (2) the interpretation of AI as a technological accelerator that reduces the cost of obtaining a working prototype and increases productivity in standard development tasks (Husein et al., 2025), while simultaneously shifting part of interface decision-making into the realm of computational support and evaluation (Brdnik et al., 2022).

Methodologically, the work combines conceptual–comparative analysis and content analysis in order to link technological shifts to changes in the economic logic of UI artefacts and development practices. First, a comparison is carried out between units of value in the monetization of component libraries and design systems before and after the mass adoption of generative techniques: from selling form and ready-made artefacts to selling manageability, guaranteed quality, and the robustness of updates. For interpretation, typologies of OSS business models and arguments about the difficulty of measuring the value of open components as a public good are employed (Duparc et al., 2022; Carter, 2024). Second, a content analysis is performed on research findings concerning which specific operations become cheaper and more reproducible (autocompletion, generation of boilerplate, acceleration of iterations) and which limitations emerge in practice (company policies, shortage of context, boundaries of trust), which makes it possible to derive the causal chain: acceleration of generation - growth of variability - increasing cost of maintaining consistent standards and verifiable contracts within design systems (Husein et al., 2025; Sergeyuk et al., 2024).

RESULTS AND DISCUSSION

Before the spread of artificial intelligence, monetization of interface component libraries was typically organized around a gap between a publicly available core and a paid safety belt for businesses: the basic version was distributed as open source software, while revenue came from extended components, visual themes, and ready-made screen sets, as well as licenses for commercial use and corporate support contracts. An additional source of income consisted of paid template packs, interface kits for design tools, and marketplaces where compatible add-ons were sold, and for large teams, software services built around the library: centralized design-system management, publication and maintenance of documentation, private component registries, and usage analytics. These trajectories fit well with the archetypal business models of open source software, where value is sold not as source code per se, but as manageability, predictability, and reduced organizational risk (Duparc et al., 2022).

Artificial intelligence disrupts this configuration primarily by devaluing scarcity: what previously required purchasing a ready-made component set or theme is increasingly reproduced through generation from textual prompts with subsequent editing, so that a component becomes a fungible commodity and a template a short-lived hint rather than a product. For interface design, methods are already being proposed that directly link textual instructions to prototype generation and adaptation of the result to user expectations, thereby reducing the premium for a pre-assembled visual appearance (Feng et al., 2025). On the development side, the same mechanism manifests as cheaper iterations: when a substantial share of boilerplate, minor fixes, and accompanying artefacts is created with the assistance of models, purchasing acceleration in the form of paid component packages begins to compete with generating a sufficiently similar solution (Husein et al., 2025).

From this emerge new expectations within teams: instead of the request to provide a component, the more frequent request becomes to assemble a screen from a textual description, but strictly within proposed design system, with high velocity in exploring alternatives and the ability to adapt the interface to the product context. Practice with programming assistants shows that developers are willing to delegate precisely routine and low-valued parts of work to the machine (in particular, tests and textual descriptions), but at the same time they report limitations of trust, organizational policies, and lack of project context; one large-scale survey gathered 481 programmer opinions, illustrating the scope and heterogeneity of expectations (Sergeyuk et al., 2024). In this configuration, money begins to flow not to the visual shell, but to guaranteed properties: accessibility for diverse user groups, performance, resilience to change, and reproducibility of outcomes across teams and versions (Zheng et al., 2024).

Thus, whereas monetization was previously anchored in selling ready-made forms and editable boilerplate, under conditions of widespread generation, this becomes a weak foundation: visual uniformity intensifies and differentiation between libraries diminishes, because models tend to reproduce the most frequent patterns and an average style. The revenue logic shifts toward what is difficult to generate in a single pass: the management of design-system rules, verifiable quality constraints, and tools that force generation to obey local product norms rather than generic templates (Duparc et al., 2022). Figure 1 presents AI Disrupts UI Component Monetization.

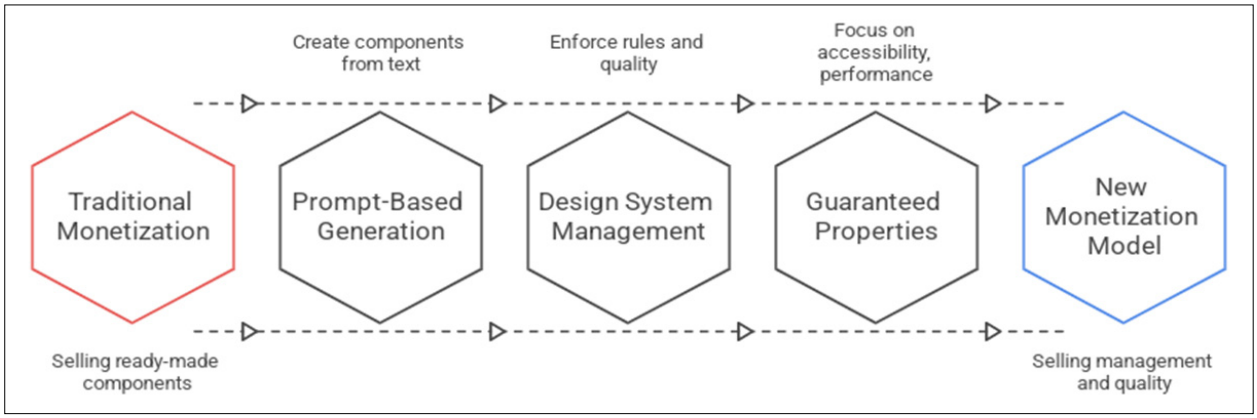

Fig. 1. AI Disrupts UI Component Monetization

In a situation where text-based generation renders standard elements almost interchangeable, the first to lose out are those who sell visual themes and simple component sets as standalone products. Their offerings relied on time savings and a ready-made look and feel, but these advantages are now easily replicated through automated generation and subsequent local tweaks, so their price premium is rapidly eroded by market expectations. Under particular pressure are libraries whose monetization is reduced to paid access to the components themselves: once a component ceases to be scarce, access to it no longer serves as a basis for a stable rent, and a substitution effect appears, whereby a team opts not to purchase but to create a sufficiently similar solution within its own constraints.

The beneficiaries are those who sell not form, but process and manageability: platforms and ecosystems that assume responsibility for orchestrating the chain of work, tool integrations, quality control, and reproducibility of outcomes. Value is not just about having an interface element, but also about integrating it into the development pipeline to automatically validate it, document it, ensure its versioning is correct, and prevent it from departing from its previous specifications as new versions are created. And in the long term, the value is not just the speed of creating variations, but ensuring the designers do not get lost in a flood of uncoordinated variations across screens.

Consequently, the unit of value itself is transformed: instead of an individual component, the system becomes central, with the component as a particular instance. Design tokens and standards move to the forefront because they define an invariant alphabet of the visual language and enable automatic verification of alignment with product norms. The role of cross-cutting integrations between design and implementation, as well as between development, testing, and release, increases, since without such a coherent loop, generation becomes a source of variability rather than a source of acceleration. In this logic, what is sold is the capacity to reconcile decisions and transfer them across tools without loss of meaning, rather than a bundle of ready-made parts.

Finally, the key commodity shifts to guarantees of quality and longevity: accessibility, security, performance, and adaptability become difficult properties to achieve with a one-generation-at-a-time approach, and require a system-wide discipline. As the number of screens and scenarios released increases, so too do the cost and the risk of behavior not working as intended, leading to a need for versioning, backward compatibility, and upgrade policies that provide predictable behavior. With the product now meeting its original use cases, the business model centers on trust in standards and controls, as well as the ability to upgrade the system without disrupting the product. Value Unit Shift is illustrated in Table 1.

Table 1. Value Unit Shift

What was sold	Why does it break under generation	What becomes the new value
Themes & ready-made aesthetics	Easy to generate and locally tweak into good enough	Standards & tokens (a stable visual alphabet)
Paid access to components	Components stop being scarce → substitution (we'll build a similar one)	System manageability (rules, docs, versioning)
Design kits as artifacts	Generation multiplies divergence faster than teams can fix it	Design-code integrations + automated checks
Speed tools (plugins, generators)	Faster creation ≠ consistent, reliable output	Quality control (a11y, regression, performance)
Library upgrades	Frequent change raises breakage risk and upgrade anxiety	Compatibility guarantees (semver, migrations, predictable upgrade paths)
Components as products	Price gets eaten as the market expects commoditized UI	Trust (reliability, security, resilience to change)

New monetization models in the era of artificial intelligence grow out of a simple observation: value shifts from owning a set of parts to managing how these parts are created, validated, and kept compatible with the system. Accordingly, an assistant increasingly forms around the library, one that does not merely generate components from descriptions, but does so within the boundaries of a specific design system, respecting tokens, composition rules, and interaction agreements. Such an assistant turns the library into a language and generation into a way of speaking it without continuous manual translation; what is monetized is not the outcome as such, but the capability to obtain the outcome in the correct form, quickly and reproducibly, with minimal entropy in code and interfaces.

The next monetization layer emerges when generation increases not only speed but also the risk of divergence: teams require paid management tools that establish a decision-making order and enforce discipline around change. This includes mechanisms for governing rules and roles, mandatory review procedures, update and deprecation policies, and control over which patterns are permissible in the product and who is authorized to modify them. In such tools, the library ceases to be a collection of files and becomes a managed organizational asset, where value is created by reducing conflicts between teams and by ensuring predictable interface evolution as the product grows.

Finally, monetization is consolidated at the level of guarantees, because guarantees are precisely what is hardest to generate within a single team: automated accessibility checks, regression comparisons of behavior and appearance, service-level requirements, and quality reporting. For large organizations, additional packages oriented toward security and control are added: deployment in closed environments, private models, activity auditing, and compliance with internal policies. A logical extension is a marketplace of certified blocks, where what is sold are not just components but components with verified properties and responsibility for their preservation, turning the add-on market from a showcase of templates into a market of trust and commitments. Monetization Models in the AI Era are shown in Figure 2.

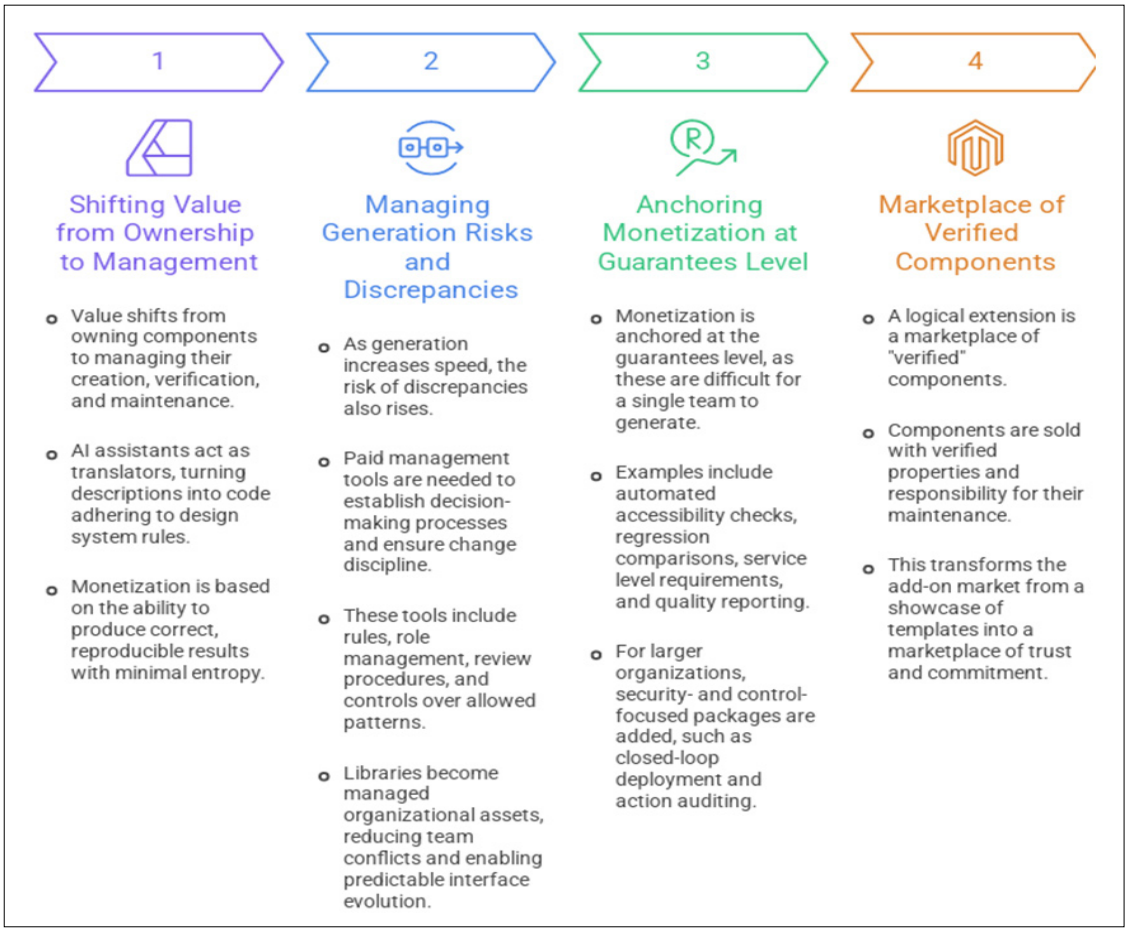

Fig. 2. Monetization Models in the AI Era

A survival strategy for interface component libraries under generative conditions begins with refusing to compete at the level of yet another button, because standard elements are the fastest to become interchangeable commodities. Libraries must focus on specialized patterns that cannot be reproduced without context, such as complex composition patterns, domain flows, conditional compositions, and temporal states. The product is not pixel-perfect, but it does have behavioral semantics and error resilience. The design language, accompanying tokens, and specification are no longer overhead: they are the product, determining the space of acceptable patterns, constraining variability, and

allowing us to tame the liberating power of generation into a controlled mutation rather than an avalanche of incompatible variations.

For a generation to function correctly, the library requires an integration layer with AI tools: extensions and interface sets that transmit local rules, component structure, and quality requirements to the models, as well as ensure traceability of why a particular result was produced. When the library becomes a source of formal constraints and verifiable contracts, it regains market power even with high form reproducibility, because it sells not appearance but correctness and compatibility. This is complemented by social infrastructure: a community, educational materials, examples, and usage practices that crystallize the system's meaning and create collective expertise. In the era of automatic generation, it is precisely such expertise that defines which solutions are considered good and keeps the library in the position of a standard rather than a disposable resource.

CONCLUSION

Artificial intelligence is being embedded into the already established infrastructure of web development, component libraries, and design systems, and thereby transforming not so much the speed of layout implementation as the economics of reusable artefacts. In an ecosystem organized as a dependency network with mass dissemination of practices through package repositories, generative and autocompleting models reduce the cost of obtaining a first working version and reproducing standard solutions, and also transfer part of design and evaluative activities into computational procedures. This erodes the former value of packaged standard solutions: ready-made themes, screen sets, and even paid access to components lose their grounding in scarcity, becoming fungible commodities in which any advantage is easily offset by generating a sufficiently similar result, which is then refined locally. Against this backdrop, the inherent tension of open components as a public good becomes more acute: as appearance and boilerplate depreciate, issues of sustainable financing, licensing, and compensation move to the center, what previously could be concealed behind the sale of forms is now forced to be articulated in terms of organizational and institutional sustainability.

Consequently, the unit of value is displaced: monetization shifts away from selling individual parts toward selling system manageability and trust in its evolution. Instead of the request provide a component, the request assemble a screen from a description, but strictly within the proposed design system intensifies, and it is precisely here that money flows to what is hard to obtain in a single generation: to tokens and standards as a stable alphabet, to design-code linkages, to automated checks, to disciplined versioning, and to predictable upgrade paths. Variants generated during this mass-divergence phase exceed teams' reconciliation capabilities. The advantage then goes to platforms and ecosystems providing quality control (accessibility, performance, security), reproducibility under change, and compatibility with external services. In the case of component libraries, the form factor matters less than defining verifiable contracts, reducing risk for organizations, and moving the add-on market from a visual one to one of guarantees, responsibility, and sustainability.

REFERENCES

1. Brdnik, S., Heričko, T., & Šumak, B. (2022). Intelligent User Interfaces and Their Evaluation: A Systematic Mapping Study. *Sensors*, *22*(15), 5830. https://doi.org/10.3390/s22155830
2. Carter, H. (2024). Measuring the Economic Value of Open Source Software. *Oxford University Press EBooks*, 743–752. https://doi.org/10.1093/oxfordhb/9780192899798.013.45
3. Duparc, E., Möller, F., Jussen, I., Stachon, M., Algac, S., & Otto, B. (2022). Archetypes of open-source business models. *Electronic Markets*, *32*(2), 727–745. https://doi.org/10.1007/s12525-022-00557-9
4. Feng, X., Du, H., Ma, J., Wang, H., Zhou, L., & Wang, M. (2025). Crafting user-centric prompts for UI generations based on Kansei engineering and a knowledge graph. *Advanced Engineering Informatics*, *65*(Part B), 103217. https://doi.org/10.1016/j.aei.2025.103217
5. Husein, R. A., Aburajouh, H., & Catal, C. (2025). Large language models for code completion: A systematic literature review. *Computer Standards & Interfaces*, *92*, 103917. https://doi.org/10.1016/j.csi.2024.103917
6. Pinckney, D., Cassano, F., Guha, A., & Bell, J. (2023). npm-follower: A Complete Dataset Tracking the NPM Ecosystem. *ArXiv*. https://doi.org/10.48550/arxiv.2308.12545
7. Sergeyuk, A., Golubev, Y., Bryksin, T., & Ahmed, I. (2024). Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology*, *178*, 107610. https://doi.org/10.1016/j.infsof.2024.107610
8. Zheng, Z., Ning, K., Zhong, Q., Chen, J., Chen, W., Guo, L., Wang, W., & Wang, Y. (2024). Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering*, *30*, 50. https://doi.org/10.1007/s10664-024-10602-0