Impact of Demographics and User Personas on GUI Development

1st Hoai Nam Ngo
Department of Computer Science
Technical University of Munich
Munich, Germany
ngothommy@gmail.com

2nd Klaudia Paździerz
Department of Computer Science
Technical University of Munich
Munich, Germany
pazdzierz.klaudia@gmail.com

3rd Nils Hothum
Department of Computer Science
Technical University of Munich
Munich, Germany
nils.hothum@tum.de

4th Alessandro Lo Muzio
Department of Computer Science
Technical University of Munich
Munich, Germany
ge42riy@tum.de

***Abstract*—This paper studies the impact of demographics and user personas on the development process of Graphical User Interfaces (GUIs), by analyzing the main areas where these insights inform decision-making, and by referencing contemporary examples and recent case studies to support the claims.**

***Index Terms*—Graphical User Interface (GUI), User-Centered Design, Personas, User Experience (UX), Software Engineering**

I. Introduction

In a market as competitive as the software market, where variables such as geolocation or scalability no longer represent major challenges, focusing on building a truly high-quality product is more crucial than ever before.

Every product, whether software or not, has three main components: **functionality**, **aesthetics** and **usability**. Together, these three factors drive user attraction and satisfaction. Software development consists of a wide range of often-overlapping activities, each with its own name and scope, with the common goal of delivering a very functional, aesthetic and usable product. One of these activities is the **Graphical User Interface development**, which focuses on the user interaction aspects by curating the aesthetics and usability parts of a software. GUI development is more than just designing a pretty interface; it spans throughout activities such as requirements analysis and visual design, up to implementation, testing and evaluation. An effective GUI represents a bridge between the functionality and the user and must be both visually appealing and highly intuitive, making it easy for users to accomplish tasks.

Netflix, for example, would likely not be as successful if a user, in order to watch a movie, had to rely on a text-based, black-and-white terminal rather than a clean, well-designed website with menus and icons.

As its name already implies, a GUI is very **user-centered**. As always in business, it is all about the user. A GUI must be designed and developed with the end user in mind. The more a GUI is tailored to it's target audience, the more effective it will be.

To personalize and tailor a GUI, it is crucial to understand who the end user is. Demographics and user personas are two powerful tools that allow to categorize and understand users on a deeper level. Demographics describe population-level statistics (e.g., age, gender, education, location), whereas personas are crafted fictional characters that embody typical users' goals, behaviours, motivations, and skill levels. Both concepts must be data-backed, behaviorally meaningful, and continuously revised to ensure they accurately reflect the target audience. Coming up with accurate demographics and personas is usually a three-step process:

- **Collect data**, e.g. through surveys, interviews, and analytics
- **Create personas based on analyzed data**
- **Validate and refine personas over time**

Once the demographics and personas are defined, they can be used to improve the GUI development process in three complementary, not mutually exclusive categories:

- **Population-Level Adaptation (Static)**,
- **Individual-Level Adaptation (Dynamic)** and
- **Process & Validation Influence**.

II. GUI Development, Demographics, and Personas

A. Graphical User Interface Development

Graphical User Interface (GUI) development encompasses the design, implementation, and evaluation of visual and interactive elements through which users interact with a system. In contrast to command-line interfaces, GUIs rely on graphical components such as windows, menus, buttons, icons, and form elements that mediate user input and system feedback. At its core, GUI development seeks to align these components with users' perceptual, cognitive, and motor capabilities, as well as with their goals and tasks in a given context of use. In the context of software systems, it is useful to distinguish between *GUI design* and *GUI development*. GUI design focuses on the conceptual and visual aspects of an interface: information

architecture, interaction flows, layout, typography, colours, and overall look-and-feel. Its goal is to define *what* the interface should communicate and *how* interaction should feel from a user's perspective.

GUI development, in contrast, comprises the engineering activities required to turn these design artefacts into a robust, interactive, and maintainable software component. It focuses on *how* the specified interface is realised in code, how it behaves under different conditions, and how it integrates with the underlying application logic and data sources.

From an engineering perspective, GUI development typically involves (i) translating user and system requirements into concrete UI behaviour and constraints, (ii) selecting suitable frameworks and architectural patterns (e.g., MVC or MVVM), (iii) implementing components, event handling, state management, and validation, and (iv) assuring quality through testing, performance optimisation, and accessibility checks. These steps are usually executed iteratively based on feedback from usage data and evaluation.

Modern GUI development is further constrained by platform guidelines and conventions (e.g., operating system human interface guidelines or web standards), which define common interaction patterns and affordances. Adhering to these conventions reduces users' learning effort and supports consistency across applications, while still leaving room for product-specific branding and interaction styles.

Importantly, GUI development is not a purely technical exercise. Implementation decisions about layout structure, interaction complexity, feedback timing, and support for assistive technologies implicitly encode assumptions about the intended user population. If these assumptions do not match the actual user base, the resulting interface may be correct from an implementation standpoint but still fail in terms of usability and accessibility.

In the following, we therefore focus on how characteristics of the target user group inform GUI design and development. We first distinguish demographics from user personas and discuss how both perspectives can be combined to derive concrete implications for graphical user interfaces.

B. User Demographics vs. Personas

Firstly it is important to distinguish these two terms, since Demographics and user personas capture two complementary perspectives on users in GUI development.

Demographics provide a population-level view that is grounded in observable facts. Attributes such as age, abilities, cultural background, and prior experience reveal systematic differences in perception, motor skills, cognitive load, and typical usage situations [1]–[5]. This evidence is essential for defining baseline requirements for usability and accessibility, for example regarding font sizes, color contrast, input modalities, or supported services. Several guidelines and standards, such as the Web Content Accessibility Guidelines (WCAG) and platform-specific human interface guidelines, explicitly build on such demographic and accessibility considerations

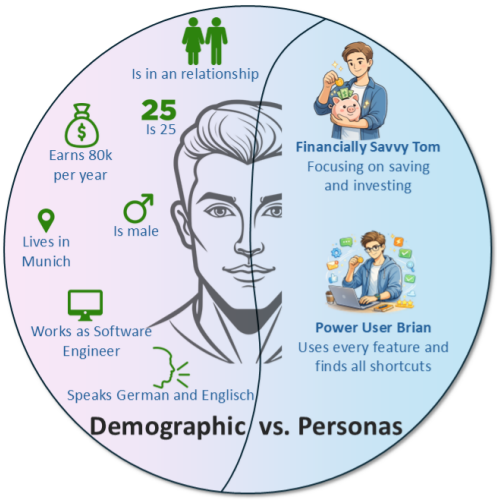

Fig. 1. Demographics provide population-level constraints, while personas capture goals, tasks, and usage context. (Grphic inspired by [TODO])

to derive concrete design recommendations. However, demographic categories like "65+" or "visually impaired" remain coarse, as they do not specify concrete goals, everyday tasks, or the actual usage context of a system.

Personas are used to translate such abstract differences into concrete user archetypes that describe typical goals, tasks, and contexts of a target group [6]. They help teams design for specific needs instead of an undefined "average user" and provide a shared reference for design decisions. However, personas are frequently criticized when they are created without empirical grounding and instead rely on stereotypes or personal assumptions [7].

Combining both perspectives mitigates these limitations. Demographic evidence can ground personas by ensuring that relevant constraints and user groups are covered, while personas operationalize demographic insights into concrete scenarios and design implications [2], [3], [8]. In this way, demographics define the breadth and diversity of the user population, and personas turn this breadth into actionable guidance for GUI design.

III. Population-Level Adaptation

A. What is Population-Level Adaptation?

Population-level adaptation refers to system-wide GUI decisions, implemented at design-time, that are based on aggregated demographic data or dominant personas, and apply to all users; for example, choosing which interface languages Netflix supports and offers to its users. Such design-time decisions are typically embedded into the system and are often quite difficult to modify once the product is deployed. Therefore, making the right assumptions about the target user population, based on the best available data, is crucial to ensure the GUI meets most users' needs.

B. Architectural Decisions: Modularity and Configurability

Building a modular system by design — in terms of code structure, application logic, and visual layout — makes it easier to adapt to new demographic or persona insights (e.g., supported languages, payment methods, sign-in options). For example, if the target demographic includes users from different countries, it is better to not hard-code the text directly into the UI, but instead to store it in external language files, which are then loaded depending on the region or the language preference of the user.

Decisions of this kind strongly affect the system's abstraction levels and testing scope. While a robust architecture demands higher initial effort to design and implement compared to a more straightforward solution, it generally pays off in the long run, by making the product easier to maintain and scale.

A widely known example of this modular approach is Netflix's microservice architecture [9]. When Netflix launched its streaming service in 2007 it adopted a monolithic architecture in which all components were tightly coupled and deployed as a single unit. In 2009, following a major outage in 2008, Netflix began migrating to a microservice architecture to address scalability and reliability issues and to facilitate their rapid expansion as a company. The platform was decomposed into many independently deployable services, increasing flexibility and simplifying future updates.

C. Country- and Region-Specific Requirements

Across all systems, the most prevalent population-level adaptation based on demographics addresses country- and region-specific requirements. Software products must adapt to different market environments. In such settings, many UI elements presented to users are subject to change, ranging from the default country language — which is the most obvious and common adjustment — to payment and sign-in options. The more a system aligns with local needs and preferences, the more successful it becomes.

A strong example of this approach is Airbnb's localization strategy [10]. Airbnb realized more than many other companies that adapting to the local market is crucial for success. One of Airbnb's local adaptations is the use of customized sign-in options that align with locally established practices; for example in the United States they include Google and Facebook, while in China they include Weibo and WeChat. Such approach helped increase Airbnb's Chinese customer base by 700% within one year. Another strategic response to anticipated market needs was Airbnb's expansion of payment methods ahead of the 2016 Rio Olympics. By extending payment options to support local Brazilian payment methods and multiple currencies beyond the U.S.-dollar, Airbnb enabled 30 million guests to book in 32 currencies and facilitated host payouts in 65 currencies.

D. Accessibility as a Baseline System Capability

A software product should be as accessible as possible to all users by design. Just as most museum entrances include wheelchair ramps beside the stairs, a GUI should be designed to accommodate as many users types and usage situations as possible, such as users with:

- **Permanent impairments**, e.g. visual, auditory, motor, cognitive impairments
- **Temporary impairments**, e.g. broken arm, wearing gloves
- **Situational constraints**, e.g. bright sunlight, noisy environments, small screens
- **Cognitive abilities**, e.g. low literacy, limited technical skills

Most solution patterns for addressing such accessibility concerns are well known and standardized in accessibility guidelines such as the Web Content Accessibility Guidelines (WCAG). The WCAG were developed by the World Wide Web Consortium (W3C), a non-profit organization dedicated to the development of open web standards and guidelines, historically hosted by institutions such as MIT and other well-known universities. The WCAG provide a technical accessability benchmark for various aspects of GUI and UX design, enabling greater inclusion of diverse personas and demographic needs. Examples include providing full keyboard navigation for users with motor impairments, ensuring sufficient color contrast for users with visual impairments, and designing interfaces that are compatible with screen readers and other assistive technologies. Despite being non-legally binding guidelines, the WCAG serve as the foundation for many legal and organizational accessibility requirements, such as the european Accessibility Act (EAA) and the U.S. Section 508 Act.

E. Key Age-Related Aspects Affecting GUI Development

1) Memory (Working and Long-Term Memory): Working memory capacity varies with age, increasing through adolescence and declining in older adulthood. Limited working memory requires GUI implementations that reduce information load, avoid long action sequences, and support recognition over recall (e.g., visible system states and saved progress). Long-term memory influences how consistently users can reuse learned interaction patterns across sessions.

2) Cognitive Processing Speed: Cognitive processing speed improves into young adulthood and gradually slows with age. GUI development must therefore account for timing constraints such as response deadlines, animations, and interaction pacing. Older users benefit from longer timeouts and systems that do not rely on rapid reactions.

3) Attention and Executive Control: Attention span and task-switching ability differ across life stages. Younger users may struggle with sustained attention, while older users may experience difficulty managing interruptions. These differences affect how developers implement multitasking, notifications, and interrupt-resilient system states.

4) Motor Abilities: Fine motor control develops throughout childhood and may decline in later life. GUI development must adjust input tolerance, gesture recognition, and sensitivity to repeated or accidental input. These considerations primarily influence event handling logic rather than visual layout.

5) Perceptual Abilities (Vision and Hearing): Sensory changes affect how users perceive system feedback. From a development perspective, this impacts feedback mechanisms, multimodal output support, and reliance on single sensory channels for conveying critical system responses.

6) Learning Ability and Experience: Younger users tend to rely more on exploratory learning, while adults draw more heavily on prior experience. GUI development must balance discoverability with consistency, ensuring that interaction logic supports both initial learning and efficient reuse.

7) Error Handling and Risk Tolerance: Age influences how users perceive and recover from errors. Children and older adults benefit from strong error prevention and simple recovery mechanisms, whereas experienced adults tolerate more complex recovery strategies. This directly affects the implementation of undo functionality, confirmation dialogs, and system safeguards.

8) Cultural Differences in Interaction Design:

F. Cultural Influences on GUI Development

1) Mental Models and User Expectations: Users from different cultures hold distinct assumptions about how systems should behave. Low-context cultures (e.g., United States, Northern Europe) expect self-explanatory interfaces and tolerate trial-and-error learning, whereas high-context cultures (e.g., Japan, South Korea) expect explicit guidance and structured onboarding.

2) Information Density and Structural Preference: Cultural background influences tolerance for information density. Some cultures prefer simplified interfaces with progressive disclosure (e.g., Germany, Scandinavia), while others are comfortable with dense layouts presenting multiple options simultaneously (e.g., China, South Korea).

3) Navigation Logic: Cultures differ in preferred navigation flow. Process-oriented cultures (e.g., Germany, Austria) favor linear, step-by-step workflows, whereas more flexible cultures (e.g., India, Southeast Asia) expect non-linear navigation and the ability to move freely between tasks.

4) Error Handling and Feedback Style: Attitudes toward errors vary culturally. In individualistic cultures (e.g., United States, Australia), errors are considered normal and direct feedback is acceptable. In collectivist cultures (e.g., Japan, China), errors are socially sensitive and feedback is often indirect and solution-focused.

5) Authority and Control (Power Distance): Power distance affects expectations of system control. Low power-distance cultures (e.g., Netherlands, Denmark) expect autonomy and customization, while high power-distance cultures (e.g., China, Russia, Arab countries) tend to trust system defaults and administrator-driven decisions.

6) Time Orientation: Cultural perception of time influences task management. Monochronic cultures (e.g., United States, Germany) favor sequential task completion and clear progress indicators, whereas polychronic cultures (e.g., Brazil, Mexico, Middle East) prefer flexible task switching and parallel workflows.

7) Language Structure Effects Beyond Translation: Language structure shapes interaction logic. Verb-oriented languages (e.g., English, German) emphasize action-based commands, while context- and noun-oriented languages (e.g., Japanese, Korean) often require object or state definition before action.

8) Privacy and Data Sensitivity Norms: Cultural norms influence privacy expectations. Some cultures demand explicit consent and transparency (e.g., Germany, EU), whereas others show higher acceptance of data collection when it enables convenience (e.g., China, South Korea). *linking to regulations in prev section*

9) Disability and Accessibility at the Individual Level:

G. Disabilities and Accessibility Needs

- Accessibility influences GUI architecture: Disabilities require GUIs to be implemented with semantic structure and accessible APIs so that interface elements can be interpreted and controlled by assistive technologies, not only visually rendered.
- Assistive technologies impose technical requirements: Screen readers and alternative input devices depend on explicit programmatic roles, states, and labels, which developers must define in the code for correct interaction.
- Keyboard accessibility affects interaction logic: Motor disabilities require full keyboard operability, influencing event handling, focus management, and navigation logic during implementation.
- Cognitive disabilities affect state and flow control: GUI logic must be predictable, avoid unexpected state changes, and provide clear error handling to support users with cognitive or neurological disabilities.
- Sensory disabilities affect data representation: Visual and hearing impairments require text alternatives, non-audio alerts, and scalable content, impacting layout behavior and notification logic.
- Performance impacts accessibility: Assistive technologies rely on stable and efficient rendering, meaning poor state management or excessive updates can reduce accessibility.
- Accessibility standards define implementation rules: Standards such as Web Content Accessibility Guidelines (WCAG) translate into concrete development requirements that are testable and enforceable.
- Accessibility reduces long-term technical debt: Integrating accessibility during development leads to modular components, cleaner logic, and improved maintainability compared to retrofitting later.

H. Scalable Layouts and Text for Diverse User Capabilities

When designing a GUI's layout and text scaling, it is essential to consider target demographics and user personas. Older users may struggle with small text or controls, while users access applications on a wide range of devices, including smartphones, tablets, and desktops. Therefore, analyzing where users come from and which devices they use is critical

to optimizing the layout. During development, fixed-size elements should be avoided in favor of relative sizing units. A responsive layout that adapts to different screen sizes, supports zooming, accommodates large text, and resizes dynamically is crucial for ensuring accessibility and broad usability.

I. Platform and Technology Choices

One of the decisions most strongly influenced by demographics and user personas is the choice of platform and technology. If the target demographic consists primarily of mobile users, as in the case of Uber, it is sensible to focus on mobile platforms (iOS, Android) and corresponding technologies rather than a web-based solution. When performance requirements are moderate, cross-platform frameworks such as React Native, Flutter, or Xamarin can reduce development effort while reaching a broad audience. If high performance is critical, native development (Swift for iOS, Kotlin for Android) is often preferable to fully exploit device capabilities. Conversely, if the target demographic mainly uses desktop systems, prioritizing desktop platforms (Windows, macOS) and technologies such as Electron or WPF may provide a richer user experience. Overall, platform and technology choices directly affect usability, performance, and scalability of the GUI.

IV. Individual-Level Adaptation

A. What is Individual-Level Adaptation?

While population-level adaptation applies to all users within a population, individual-level adaptation refers to GUI adaptations tailored to a single user. Such adaptations can occur either automatically, imposed by the system (**Personalization**), or be explicitly enabled by the user (**Customization**).

The decision to implement any form of customization is typically driven by demographic and persona heterogeneity and therefore constitutes a relevant development concern. However, since this form of customization is optional and enhances the UI/UX only when actively discovered and used by the relevant user groups, it does not require the same level of demographic and persona analysis as personalization, which addresses more relevant UI enhancements that are applied automatically by the system.

B. When to use Customization?

Assuming a company has a clear and reliable understanding of the needs of a specific niche user group, the optimal approach is to implement personalization automatically within the UI/UX rather than relegating such improvements to deeply buried configuration settings that rely on users actively discovering and actively enabling them. However, given the substantial cost of false positives in such cases, implementing personalization requires strongly data-backed inferences, which can rarely be guaranteed with enough certainty. For this reason, customization is widely used in the industry, especially in applications that aim to satisfy diverse user needs.

Mobile applications, for example, tend to offer less customization than other platforms and place a stronger emphasis on simplicity and task-oriented UX. Users of such applications are typically more focused on accomplishing tasks than on customizing the interface, a tendency that is partly influenced by limited screen size. [11]

However, in some cases — such as the Yazio calorie-tracking app — the mobile application depends heavily on the user persona and must therefore require users to complete an initial customization process, such as a multiple-choice quiz.

V. Process and Validation Influence

A. Persona-Informed Design Processes

B. Validation and Evaluation Methods

C. Adaptation Challenges

VI. Impact of Artificial Intelligence in GUI Development

A. AI-Assisted Development for Persona- and Demographic-Aware GUIs

- **Embedding personas and demographics into prompts:**
  - AI coding assistants can be guided with explicit information about target personas (e.g., novice vs. expert users) and demographic constraints (e.g., older adults, users with visual impairments).
  - When prompts include these aspects, generated GUI code is more likely to reflect appropriate interaction complexity, font sizes, contrast, or navigation depth for the intended user groups.
- **Operationalizing persona and demographic requirements in code:**
  - AI can help translate high-level persona descriptions (goals, tasks, context) into concrete UI elements and interaction flows (e.g., simplified wizards for infrequent users, shortcut-heavy views for experts).
  - Demographic evidence (e.g., accessibility guidelines, age-related constraints) can be referenced explicitly in prompts so that the assistant proposes components that follow these constraints.
- **Risk of generic, persona-blind suggestions:**
  - Without explicit persona and demographic context, AI defaults to generic patterns that may primarily fit "average" or highly experienced users.
  - This can unintentionally marginalize critical user segments, such as older users or people with disabilities, even if personas for these groups exist in the project.

B. Vibe Coding and the Role of Personas and Demographics

- **Vibe coding with persona-aware prompts:**
  - During exploratory "vibe coding", developers can incorporate persona labels (e.g., "design this view for an anxious first-time user") and demographic cues (e.g., "suitable for low-vision users on mobile") directly in natural-language prompts.

  - This keeps rapid prototyping aligned with the defined user models instead of drifting towards purely aesthetic or developer-centric preferences.
- **Danger of drifting away from user models:**
  - If vibe coding relies only on generic prompts like "modern dashboard" or "clean settings page", AI-generated GUIs may ignore the specific needs of the personas and demographic groups identified earlier in the project.
  - Such drift makes it harder to justify design decisions with respect to user diversity and may contradict accessibility and inclusion goals.

C. Limitations of AI for Persona- and Demographic-Sensitive GUIs

- **Lack of embedded user models:**
  - Current AI tools do not maintain an internal, project-specific model of personas or demographic segments; they only react to what is stated in the prompt or visible in the code.
  - As a result, persona and demographic considerations can easily be lost once they are not explicitly mentioned in each interaction.
- **Inconsistent alignment with guidelines:**
  - Even when accessibility or demographic constraints are mentioned, AI suggestions may only partially follow relevant guidelines (e.g., WCAG) and still require manual review against persona and demographic requirements.
  - Over-reliance on AI-generated code increases the risk that subtle needs of certain personas (e.g., low digital literacy, anxiety, motor limitations) are overlooked.

VII. Conclusion

Section reference VII

Acknowledgment

We would like to thank Sidong Feng for supervising this project and providing valuable guidance on Graphical User Interface design. We also thank the Chair of Software Engineering & AI at TUM for providing course resources and support.

References

[1] W. W. W. Consortium *et al.*, "Web content accessibility guidelines (wcag) 2.0," 2008.

[2] J. S. Rodríguez-Cardiel, H. Luna-García, J. M. Celaya-Padilla, L. M. Zapata-Alvarado, J. Ramírez-Carrillo, J. F. Hernández-Serrano, and L. E. Salcedo-Bugarín, "Prototype user interface for an automatic sheep weighing control system: Implementing iso 9241-210: 2019, iso 29110, ieee 830." in *HCI-COLLAB*, 2022, pp. 83–93.

[3] T. Clemmensen, M. Hertzum, K. Hornbæk, Q. Shi, and P. Yammiyavar, "Cultural cognition in usability evaluation," *Interacting with computers*, vol. 21, no. 3, pp. 212–220, 2009.

[4] R. Atata and T. Odedeyi, "Designing inclusive interfaces: Enhancing user experience for people with disabilities through adaptive ui accessibility principles," 2025.

[5] J. Salminen, K. Guan, S.-G. Jung, and B. J. Jansen, "A survey of 15 years of data-driven persona development," *International Journal of Human–Computer Interaction*, vol. 37, no. 18, pp. 1685–1708, 2021.

[6] M. Gomez-Hernandez, X. Ferre, C. Moral, and E. Villalba-Mora, "Design guidelines of mobile apps for older adults: systematic review and thematic analysis," *JMIR mHealth and uHealth*, vol. 11, p. e43186, 2023.

[7] N. Wagner, K. Hassanein, and M. Head, "The impact of age on website usability," *Computers in Human Behavior*, vol. 37, pp. 270–282, 2014.

[8] T. W. Howard, "Are personas really usable?" *Communication Design Quarterly Review*, vol. 3, no. 2, pp. 20–26, 2015.

[9] N. Tariq. (2025, September) Netflix architecture case study: How does the world's largest streamer build for scale? Clustox. Accessed: 2026-01-23. [Online]. Available: https://www.clustox.com/blog/netflix-case-study/

[10] United Language Group. (2017) Airbnb's localization strategy. Originally published August 2017; updated version. [Online]. Available: https://www.unitedlanguagegroup.com/blog/airbnb-localization-strategy

[11] M. Apostol. (2024, June) 12 ux best practises for mobile apps. HyperSense Software. Accessed: 2026-01-26. [Online]. Available: https://hypersense-software.com/blog/2024/06/19/12-ux-best-practices-for-mobile-apps/

[12] J. Smith and J. Doe, "An example study on latex," *Journal of Examples*, vol. 10, no. 2, pp. 12–34, 2020.

[13] J. Pruitt and J. Grudin, "Personas: practice and theory," in *Proceedings of the 2003 conference on Designing for user experiences*, 2003, pp. 1–15.