qaware.de

se.uni-bayreuth.de

UNIVERSITÄT BAYREUTH

QA|WARE
SOFTWARE ENGINEERING

UX Debt: Developers Borrow While Users Pay

Sebastian Baltes, Veronika Dashuber

sebastian.baltes@uni-bayreuth.de, veronika.dashuber@qaware.de

How Do We Define UX Debt?

Software developers borrow from the **maintainability** and **extensibility** of a software system for a short-term speed up in development time.

Developers are the ones who **pay the interest** in form of longer development times.

Tech Debt

vs.

Software developers borrow from the end **users' experience** and **productivity** for a short-term speed up in development time.

They are not the ones who **pay the interest**, the **actual users** are.

UX Debt

Methodology: Determine UX Debt

1

Case study based on our industry experience

- Definition of three UX debt classes
- Informal validation with colleagues working on other projects

2

Detailed company-internal online survey

- Conducted with professional software engineers
- Gauging their perspectives on the classes we proposed

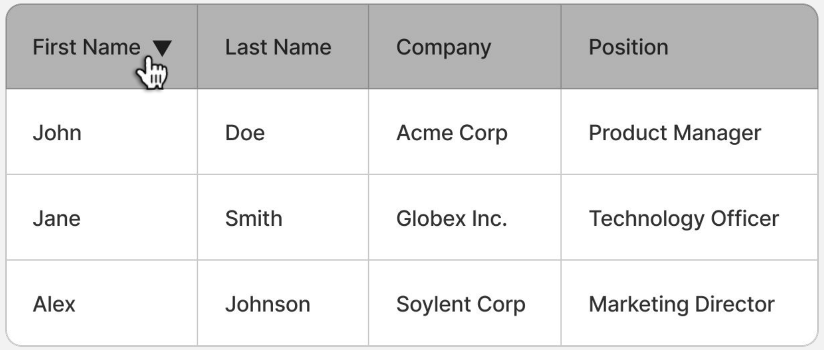
QA|WARE

UX Debt Classes

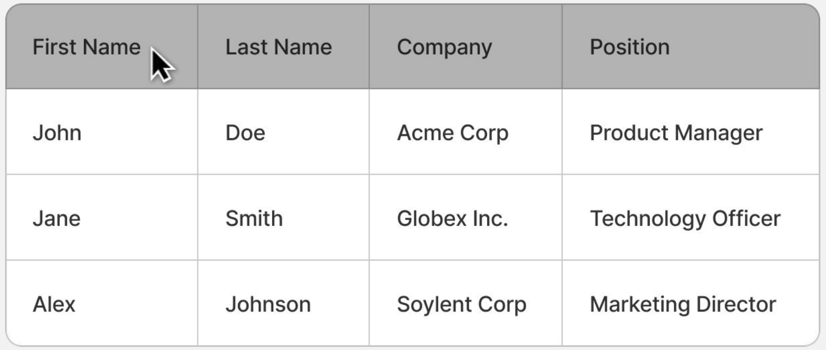
Code-centric UX Debt

Context	Traditional code-centric **technical debt** that **causes usability issues**.
Problem	The debt that **users pay** due to the reduced usability of the application is not factored in.
Example	Code clones in CSS classes that cause **inconsistent** and thus irritating user-facing **behavior** of front-end components
Mitigation	Configure **static analysis tools** to scan CSS, HTML, and other UI-focused files

Code-centric UX Debt



First Name ▼	Last Name	Company	Position
John	Doe	Acme Corp	Product Manager
Jane	Smith	Globex Inc.	Technology Officer
Alex	Johnson	Soylent Corp	Marketing Director



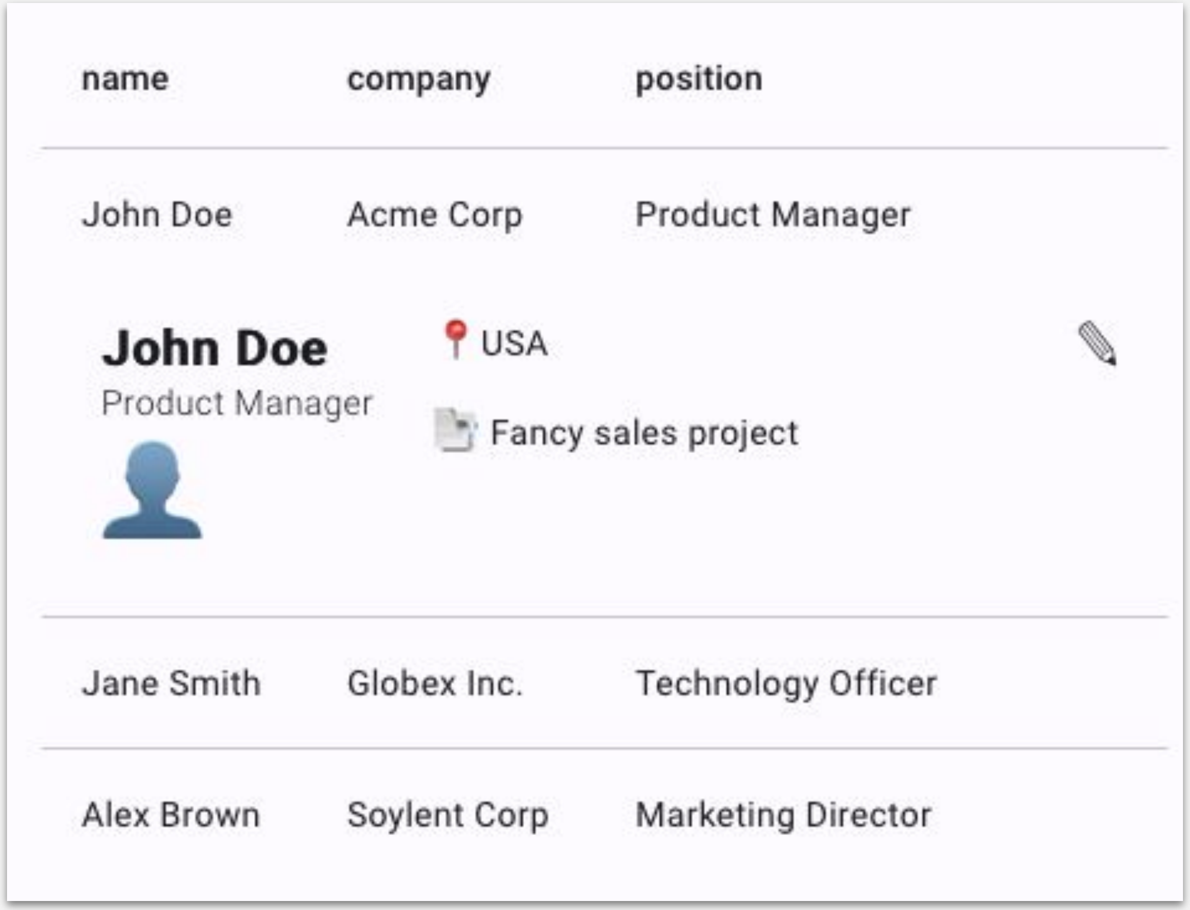
First Name	Last Name	Company	Position
John	Doe	Acme Corp	Product Manager
Jane	Smith	Globex Inc.	Technology Officer
Alex	Johnson	Soylent Corp	Marketing Director

Missing filter indicator due to code duplications

Architecture-centric UX Debt

Context	**Suboptimal placement of components** implied by architectural decisions.
Problem	Architecture decisions are made **without taking into account** how to arrange **UI** components or their coherence across views.
Example	A certain **API design** or component split is technically more convenient but it **leads to suboptimal** placement for the **UX**.
Mitigation	Besides code review, add an additional **domain review** step in which a domain expert evaluates features from a UI/UX perspective, providing early feedback.

Architecture-centric UX Debt



name	company	position
John Doe	Acme Corp	Product Manager

John Doe
Product Manager

📍 USA

Fancy sales project

name	company	position
Jane Smith	Globex Inc.	Technology Officer
Alex Brown	Soylent Corp	Marketing Director

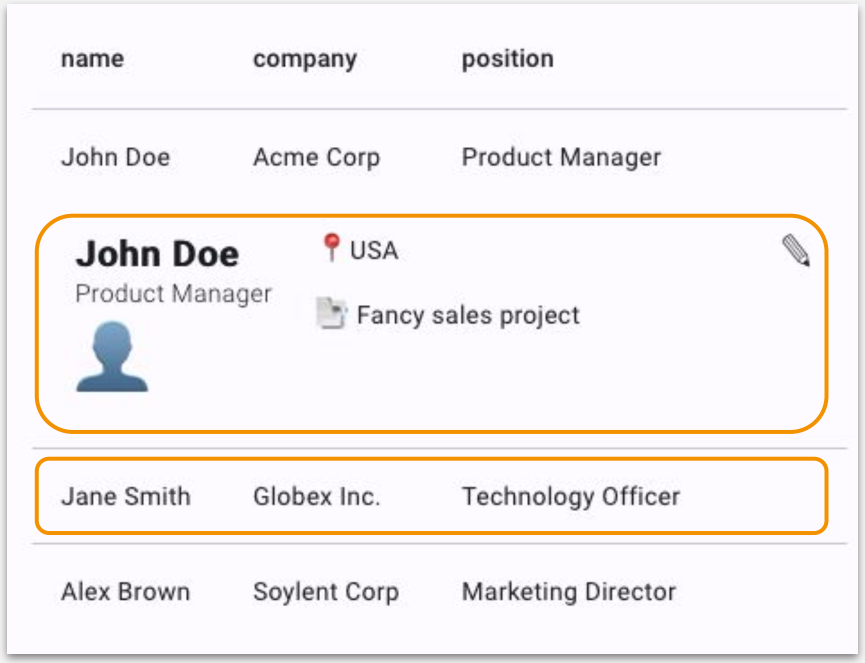
Architecture-centric UX Debt

Employee Details Object

```
{
 "id": 1,
 "name": "John Doe",
 "company": "Acme Corp",
 "position": "Product Manager",
 "location": "USA",
 "project": "Fancy sales project",
 "imageUrl": "photo.jpeg"
}
```

Table Details Component

Table Row Component



name	company	position
John Doe	Acme Corp	Product Manager

John Doe
Product Manager
USA
Fancy sales project

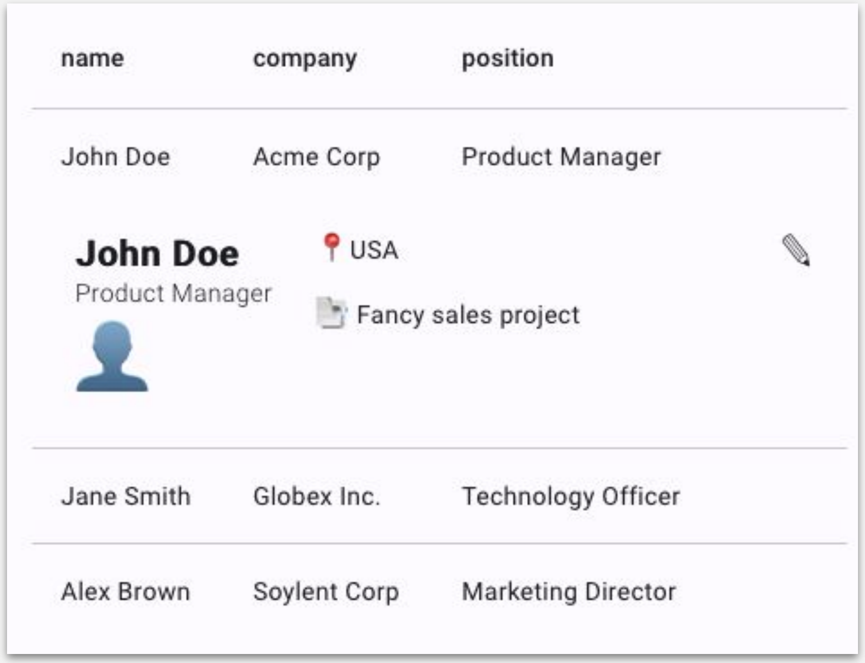
Jane Smith	Globex Inc.	Technology Officer
Alex Brown	Soylent Corp	Marketing Director

Employee Object

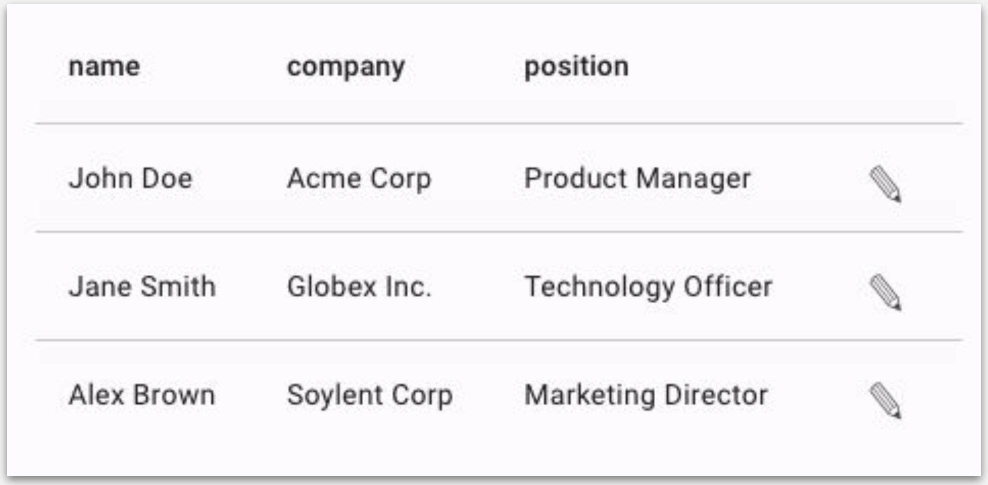
```
{
 "id": 1,
 "name": "John Doe",
 "company": "Acme Corp",
 "position": "Product Manager"
}
```

```
public void updateEmployee(EmployeeDetails employee) {
   // ...
}
```

Architecture-centric UX Debt



name	company	position
John Doe	Acme Corp	Product Manager
Jane Smith	Globex Inc.	Technology Officer
Alex Brown	Soylent Corp	Marketing Director



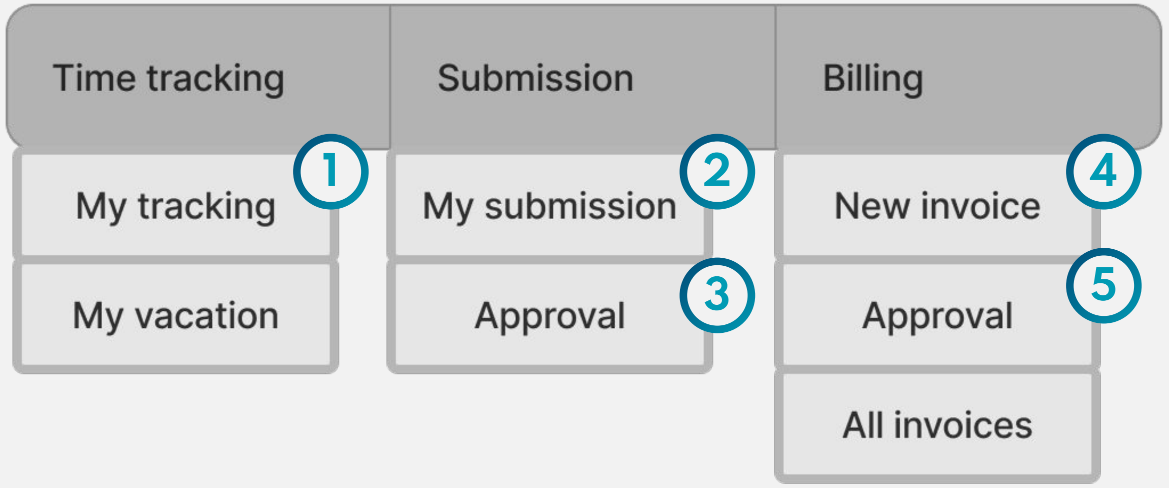
name	company	position
John Doe	Acme Corp	Product Manager
Jane Smith	Globex Inc.	Technology Officer
Alex Brown	Soylent Corp	Marketing Director

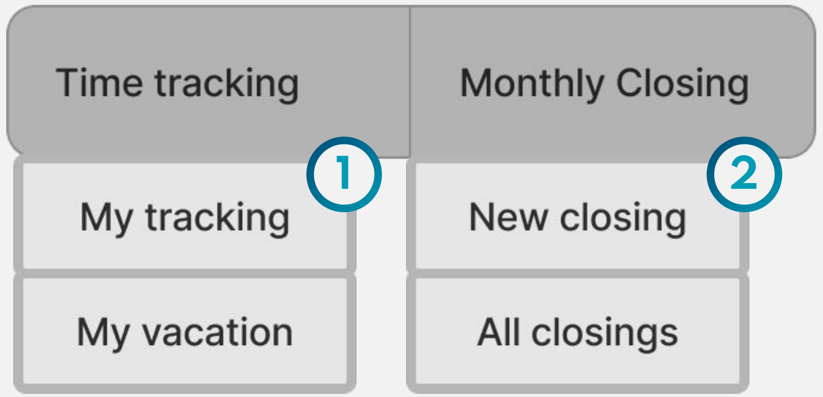
Back-end/Front-end Misalignment:
Edit is only possible in the details and not in the table itself (would be quicker for the user) because the details object is needed for the update endpoint

Process-centric UX Debt

Context	The application as a whole **does not support the users**' desired and most efficient ways of **working**.
Problem	Users' workflows are not taken into account during development; the **application cannot be used intuitively and efficiently**.
Example	A common **user workflow** is **spread over** several UI pages of the **application**.
Mitigation	Conduct **user research**, before and during implementation using interviews, surveys, or log data analysis.

Process-centric UX Debt





The process of monthly closing is spread across 5 pages in the menu on the left.

TLDR;

It might be that not only the developers pay for the technical debt they have built up, but also the end users.

There are tradeoffs between a for developers clean and convenient architecture and low UX.

A completely tech-debt-free software can still have a crappy user experience.

Q&A

Paper Preprint

Sebastian Baltes, Veronika Dashuber

sebastian.baltes@uni-bayreuth.de, veronika.dashuber@qaware.de