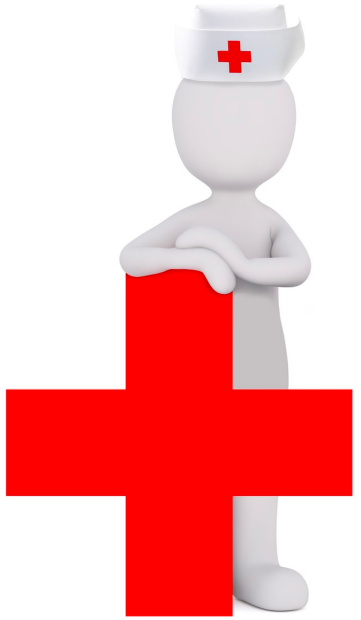
LinkedIn: https://www.linkedin.com/in/marcin-k%C4%99pka-74060a176/

CHARAKTERYSTYKA



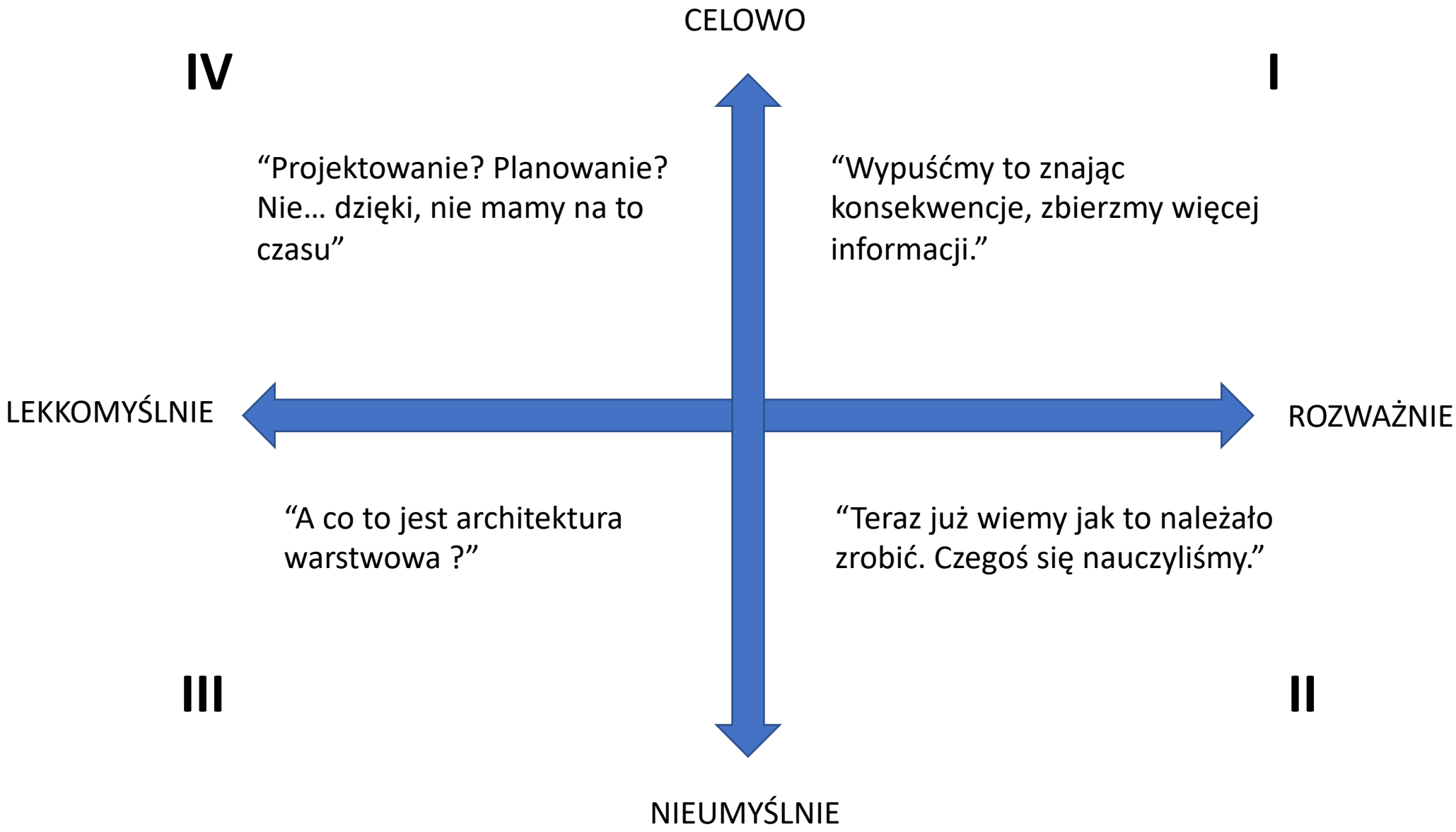
Ward Cunningham

“Wypuszczanie kodu po raz pierwszy przypomina zaciąganie długu. Trochę długu przyśpiesza dewelopment o ile spłacamy go szybko poprzez przepisanie kodu… Niebezpieczeństwo pojawia się kiedy dług nie jest spłacany. Każda minuta spędzona z “nie do końca właściwym kodem” liczy się jako odsetki od tego długu.”

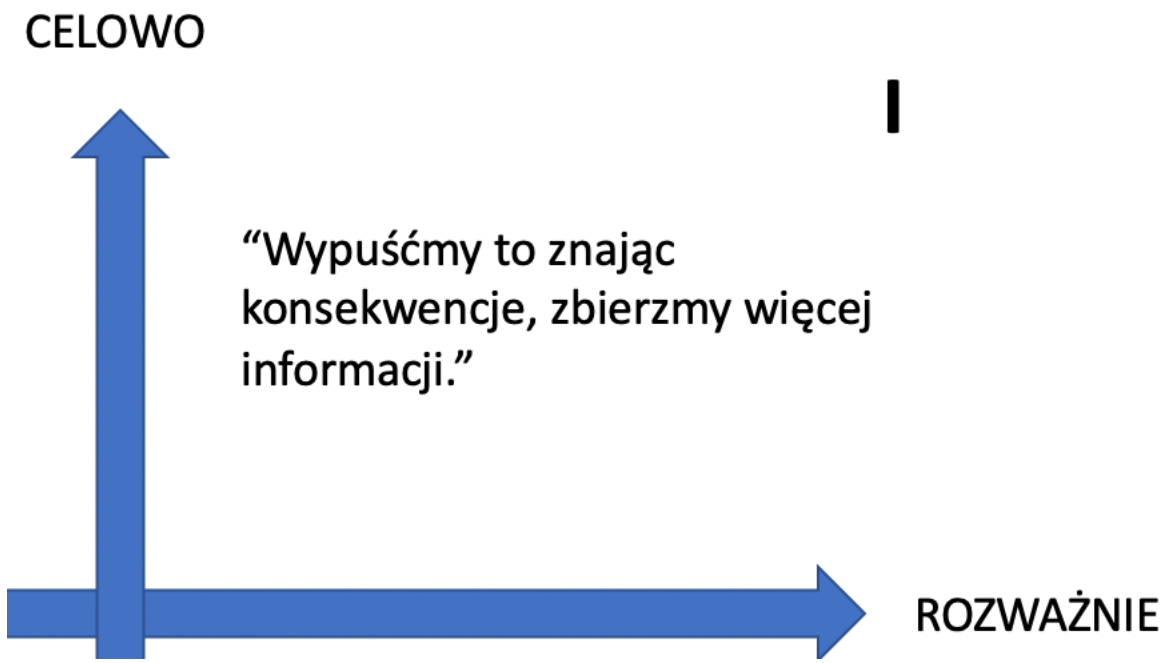
"The WyCash Portfolio Management System". OOPSLA conference, 1992.

Dług techniczny – koncepcja, która w inżynierii oprogramowania odzwierciedla ukryty koszt dodatkowych przeróbek, które są wymagane z powodu wybrania łatwego i szybkiego rozwiązania zamiast właściwego, które wymagałoby więcej czasu.

Kwadrant długu technicznego



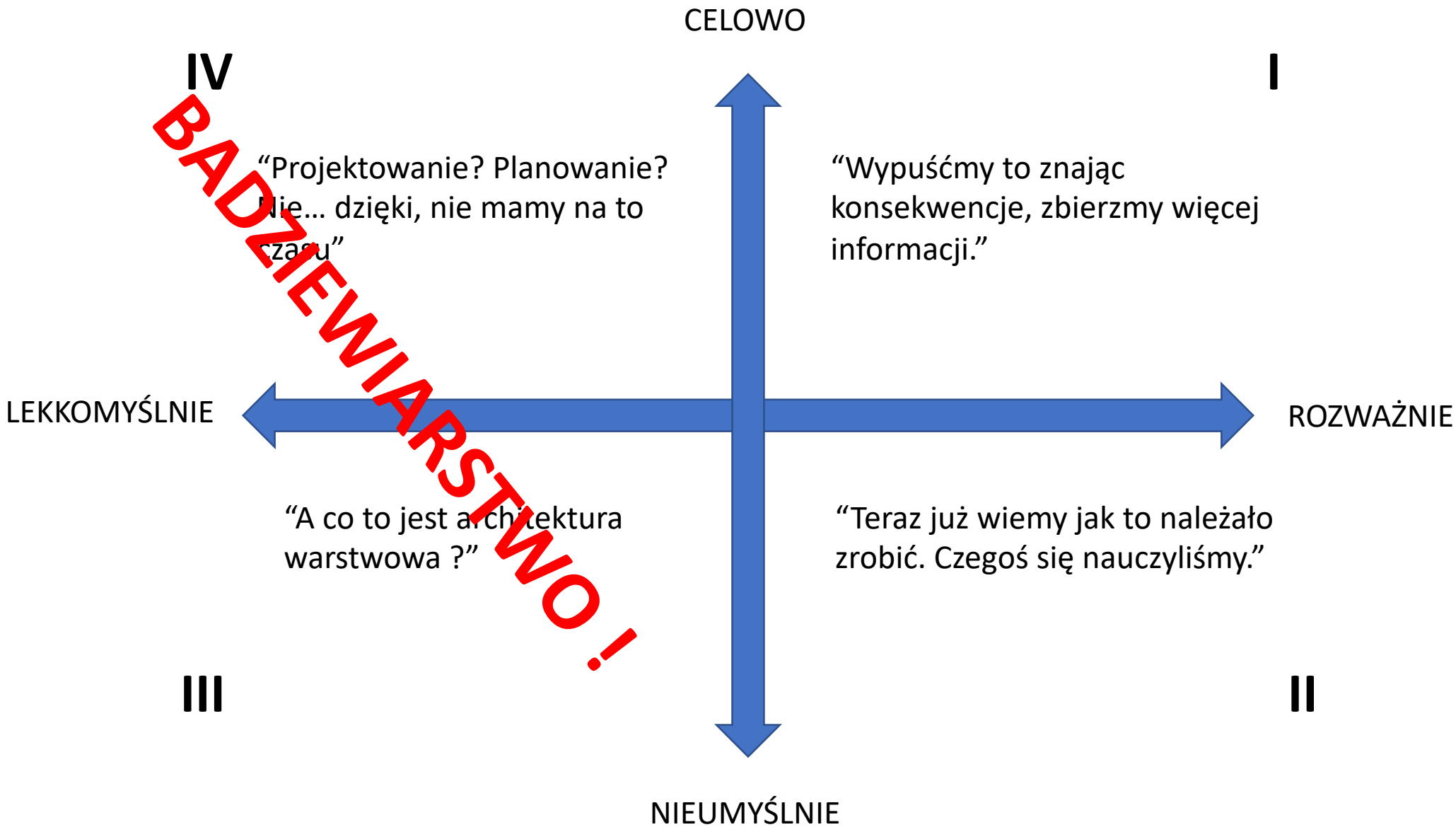
Źródło: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html



Czy mądrze zaciągamy dług techniczny ?

- ❑ Czy jest szansa że się czegoś nauczymy i wpłynie to na finalny kształt kodu ?
- ❑ Czy jest plan na spłacenie długu ?
- ❑ Czy biznes jest NAPRAWDĘ poinformowany ?
- ❑ Czy kod jest czysty ?
- ❑ Czy kod jest przetestowany ?

Kwadrant długu technicznego



Źródło: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[Scrum Team] to spójna grupa **PROFESJONALISTÓW** skupionych na pojedynczym celu określanym jako Cel Produktu.

Scrum Guide 2020, s. 5

PRZYCZYNY

Przyczyny powstawania długu technicznego:

1. Kod – napisany w niewłaściwy sposób, niepotrzebnie duplikowany, złożony, ciężki do zrozumienia i zmiany w przyszłości
2. Projekt i architektura – dług jest wynikiem nieoptymalnych pierwotnych założeń lub rozwiązań, które wraz z rozwojem technologii stają się nieoptymalne
3. Środowisko – dług związany z otoczeniem aplikacji, czyli procesy związane z wytwarzaniem oprogramowania (np. brak CI/CD), sprzęt (hardware), infrastruktura i aplikacje wspierające
4. Dystrybucja wiedzy – dług wynika z braku właściwego zarządzania wiedzą, np. brak dzielenia się wiedzą w zespole, brak aktualnej dokumentacji
5. Testowanie – wynika np. z braku skryptów testowych (przez co konieczne jest ręczne testowanie aplikacji przed każdym wydaniem) lub niewystarczającego pokrycia aplikacji testami (ręcznymi lub automatycznymi)

Źródło: E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt,"Journal of Systems and Software, vol. 86, no. 6, pp. 1498–1516, 2013.

PROFILATKYKA

PROFILAKTYKA

- Kiedy zaciągamy dług techniczny musi on się znaleźć w backlogu produktu (zrozumiały opis, wycena, uzgodnienie z PO w zakresie planu “spłaty”)
- Zasady, pryncypia projektowania rozwiązań
- Zbiór dobrych praktyk kodowania w danym języku (budowanie wiedzy)
- Transparentna jakość projektowania i planowania (ilość zależności pomiędzy klasami, obiektami w kodzie, diagramy)
- Dobre, poddawane inspekcji i adaptacji DoD
- Utrzymanie aplikacji jest częścią dewelopmentu
- Nigdy nie rób świadomie bałaganu, to nie ma nic wspólnego z długiem technicznym
- Refactoring musi być stałą tradycją, częścią procesu wytwórczego
- Nie rób co jakiś czas sprintu na refactoring

- Zawsze zostawiaj po sobie czysty i zrozumiały dla innych kod (trochę taka harcerska zasada - kiedy opuszczasz obóz zostaw go czystym)
- Nigdy nie pytaj nikogo o pozwolenie na właściwe wykonanie swojej pracy (jakość jest twoim obowiązkiem!)
- Automatyzacja testów
- TDD
- Monitoruj wygasające technologie (np. Adobe Flash Player, Cobol)
- Zarządzaj wiedzą i wymianą doświadczeń (czas na wdrożenie nowych osób, gildie, repozytoria wiedzy, dokumentacja…)
- Ustal kto odpowiada za architekturę i standardy projektowania rozwiązań
- Zwracaj uwagę na wymagania niefunkcjonalne
- Wycofuj martwy kod lub całe funkcjonalności, jeśli nie cieszą się uznaniem użytkowników
- Monitoruj dług techniczny, mierz jego wielkość

DIAGNOSTYKA

Propozycja mierników

- Wycena pracy potrzebnej do wykonania w momencie zaciągania długu technicznego

- Technical Debt Ratio [TDR] = $\frac{\text{koszt pracy potrzebnej do spłacenia długu}}{\text{koszt zbudowania aplikacji}}$

- Pokrycie kodu testami (%) - manualne lub automatyczne

- Pokrycie kodu testami automatycznymi (%)

- Ilość wycofanego kodu z produkcji w ustalonej jednostce czasu (liczone np. liniami kodu)

- Wymyślcie kilka historyjek, które mogłyby ale pewnie nigdy nie zostaną wdrożone i co jakiś czas estymujcie koszt ich wdrożenia. Jeżeli rośnie, to znaczy że kod naszego oprogramowania się pogarsza.

- Statystyki niepoprawionych błędów

- Time-to-market

- Ilość kodu pokryta dokumentacją

OBJAWY I LECZENIE

Czy w moim produkcie jest dług techniczny wymagający działania ?

Odpowiedz sobie na te pytania:

- ❑ Czy rozwijanie oprogramowania jest spowolnione ?
- ❑ Czy zdarzają się częściowe lub całościowe przestoje w działaniu systemu ?
- ❑ Czy powracają te same błędy ?
- ❑ Czy czas wdrażania nowych funkcjonalności stale wzrasta ?
- ❑ Czy aplikacja działa wolno ?
- ❑ Czy twoi programiści niechętnie pracują nad produktemt ?
- ❑ Czy nowe osoby w zespole nie są w stanie zapoznać się z aplikacją korzystając z dokumentacji ?
- ❑ Czy programiści nie dzielą się regularnie wiedzą (CoP, gildia,....) ?
- ❑ Czy jest w organizacji presja na wdrażanie nowych funkcjonalności szybciej, niż zespół estymuje ?
- ❑ Czy zdarzają się błędy trudne do odtworzenia lub rozwiązania ?
- ❑ Czy są takie miejsca w aplikacji, w których żaden programista nie chce robić zmian ?
- ❑ Czy słychać coraz częściej z zespołu głosy “napiszmy system od nowa” ?
- ❑ Czy dług techniczny był do tej pory lekceważony (brak świadomego zarządzania, brak zgody biznesu na “robienie technicznych rzeczy w sprincie”) ?

Strategie wyjścia z dużego długu technicznego:

- przebudowa/unowocześnienie poprzez refaktoryzację kodu,
- całościowa wymiana systemu na nowy,
- stopniowe przepisanie aplikacji (rewrite - napisanie od nowa kodu danej funkcjonalności, bez analizy i poprawiania starego kodu)


MISTRZU, JAK DŁUGO TRZEBA CZEKAĆ, BY COŚ SIĘ ZMIENIŁO NA LEPSZE?
JEŻELI BĘDZIESZ CZEKAŁ, TO BARDZO DŁUGO...

Q&A