Managing Technical Debt

Doug Klugh

Software Engineering Leader

Doug Klugh

- ✓ B.S. in Computer Science (Software Engineering)
- ✓ 36 years delivering firmware/software for some of the world's top brands
- ✓ 20+ years of practice leadership spanning all SDLC disciplines
- ✓ 15+ years of Agile leadership
- ✓ Delivered consumer & enterprise software servicing > one billion users

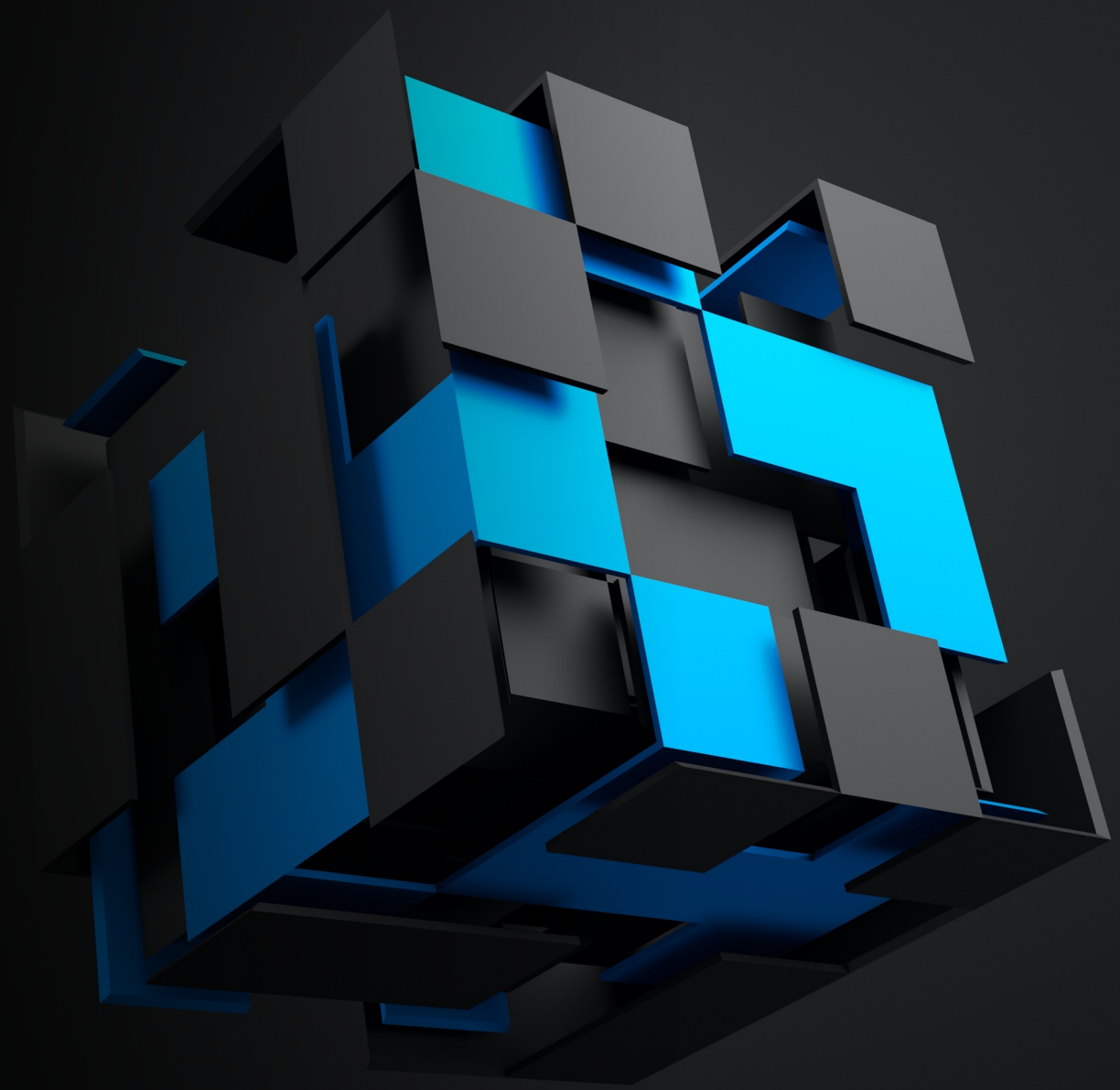
Agenda

- What is Technical Debt?
- Analogy to Financial Debt
- Types of Technical Debt
- Reasons for Technical Debt
- Realizing Technical Debt
- Leveraging Technical Debt

> A deficiency in the expected level of quality for individual artifacts or artifacts integrated as a whole.

Technical Debt

Quality Model

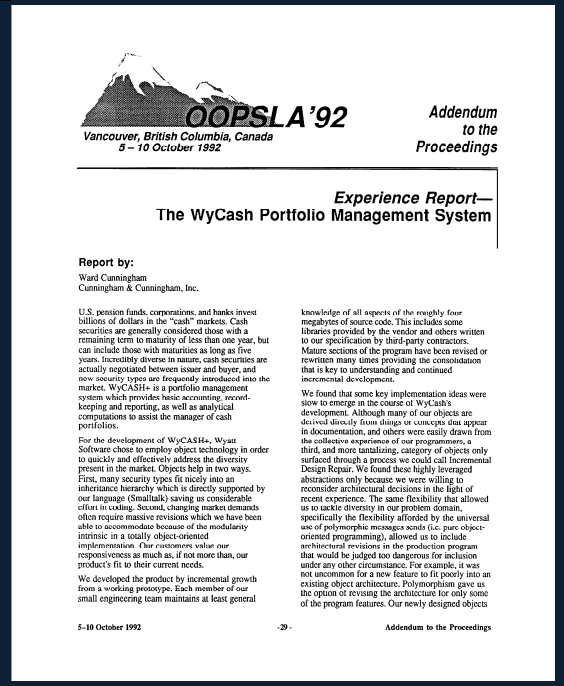


Ward Cunningham

Shipping first time code is like going into debt.

A little debt speeds development so long as it is paid back promptly with a rewrite.

An Analogy to Financial Debt



OOPSLA'92
Vancouver, British Columbia, Canada
5 – 10 October 1992

Addendum to the Proceedings

Experience Report—
The WyCash Portfolio Management System

Report by:
Ward Cunningham
Cunningham & Cunningham, Inc.

U.S. pension funds, corporations, and banks invest billions of dollars in the "cash" markets. Cash securities are generally considered those with a remaining term to maturity of less than one year, but can include those with maturities as long as five years. Incredibly diverse in nature, cash securities are actually negotiated between issuer and buyer, and new security types are frequently introduced into the market. WyCASH+ is a portfolio management system which provides basic accounting, record-keeping and reporting, as well as analytical computations to assist the manager of cash portfolios.

For the development of WyCASH+, Wyatt Software chose to employ object technology in order to quickly and effectively address the diversity present in the market. Objects help in two ways. First, many security types fit nicely into an inheritance hierarchy which is directly supported by our language (Smalltalk) saving us considerable effort in coding. Second, changing market demands often require massive revisions which we have been able to accommodate because of the modularity intrinsic in a totally object-oriented implementation. Our customers value our responsiveness as much as, if not more than, our product's fit to their current needs.

We developed the product by incremental growth from a working prototype. Each member of our small engineering team maintains at least general knowledge of all aspects of the roughly four megabytes of source code. This includes some libraries provided by the vendor and others written to our specification by third-party contractors. Mature sections of the program have been revised or rewritten many times providing the consolidation that is key to understanding and continued incremental development.

We found that some key implementation ideas were slow to emerge in the course of WyCash's development. Although many of our objects are derived directly from things or concepts that appear in documentation, and others were easily drawn from the collective experience of our programmers, a third, and more tantalizing, category of objects only surfaced through a process we could call Incremental Design Repair. We found these highly leveraged abstractions only because we were willing to reconsider architectural decisions in the light of recent experience. The same flexibility that allowed us to tackle diversity in our problem domain, specifically the flexibility afforded by the universal use of polymorphic messages sends (i.e. pure object-oriented programming), allowed us to include architectural revisions in the production program that would be judged too dangerous for inclusion under any other circumstance. For example, it was not uncommon for a new feature to fit poorly into an existing object architecture. Polymorphism gave us the option of revising the architecture for only some of the program features. Our newly designed objects

Assuming Debt

Interest Payments

Principle Payments

Types of Technical Debt

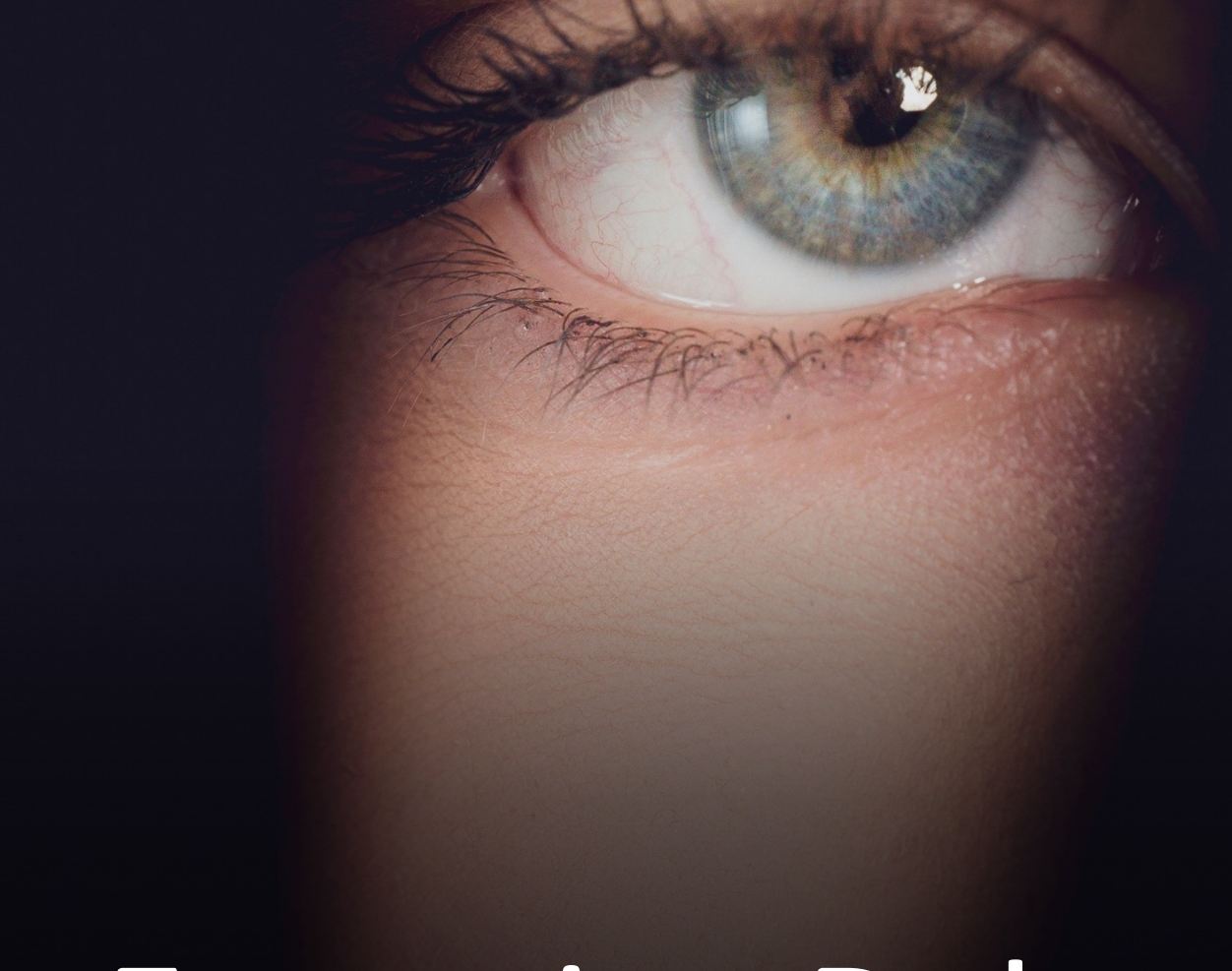
- Architectural Debt
- Code Debt
- Test Debt
- Knowledge Debt
- Emerging Debt

Architectural Debt

Code Debt

Test Debt

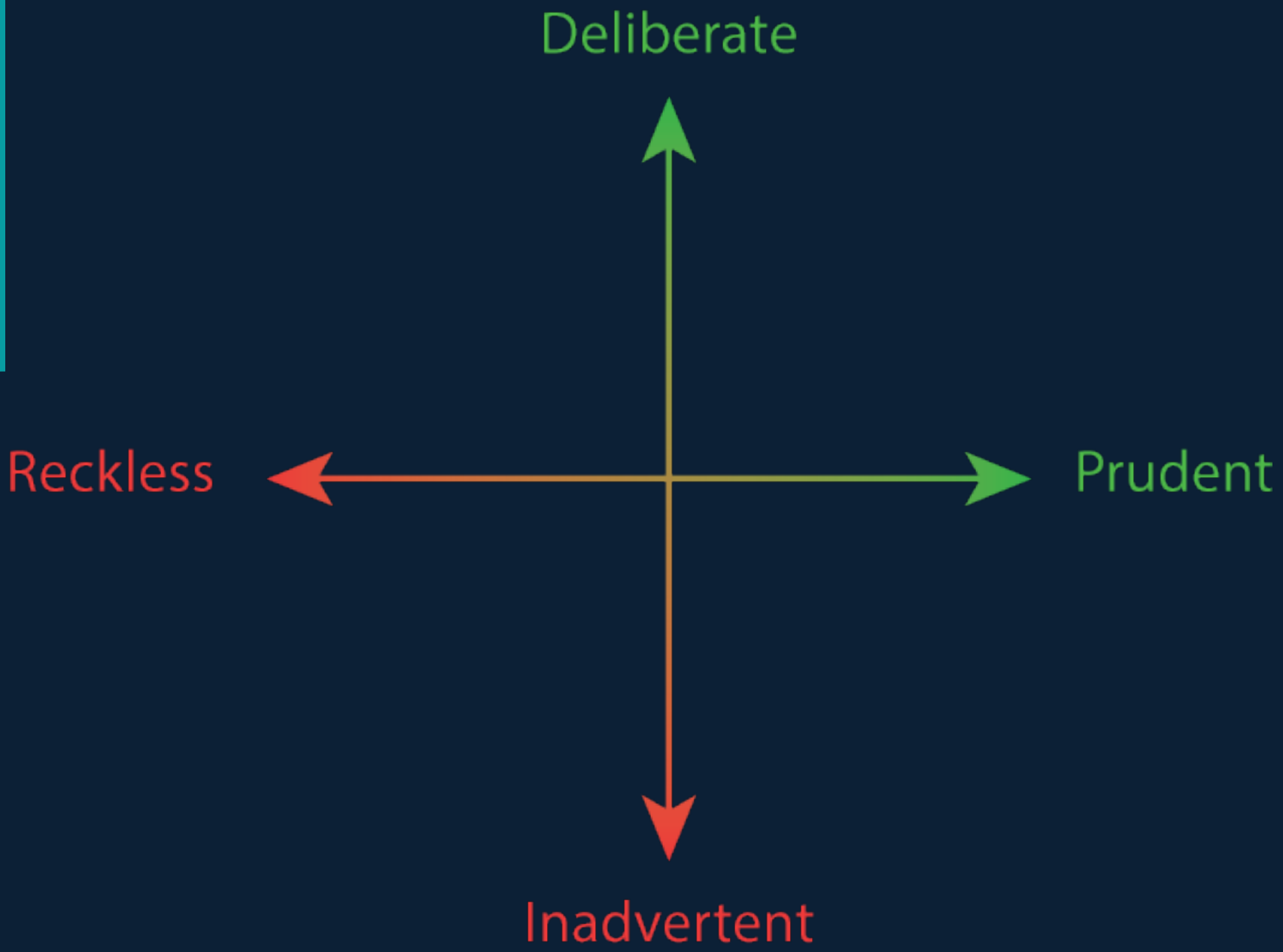
Knowledge Debt



Emerging Debt

Reasons for Technical Debt

Technical Debt Quadrant



Realizing Technical Debt

> Make technical debt visible to all stakeholders

Track

Technical Debt

> Determine impact
> to quality model

Analyze

Technical Debt

> Quantify effort,
> complexity, and risk

Size

Technical Debt

> Ensure technical debt
> is paid off quickly

Prioritize

Technical Debt

devLead.io

Leveraging Technical Debt

Ward Cunningham
@WardCunningham

Dirty code is to technical debt as the pawn broker is to financial debt. Don't think you are ever going to get your code back.

2:51 PM · Sep 3, 2009

Thank You

This deck is available at:

www.github.com/Klugh/Course-ManagingTechnicalDebt

Follow me to learn more

Doug Klugh
doug@DevLead.io
/Klugh
@DougKlugh
/Klugh
/Klugh