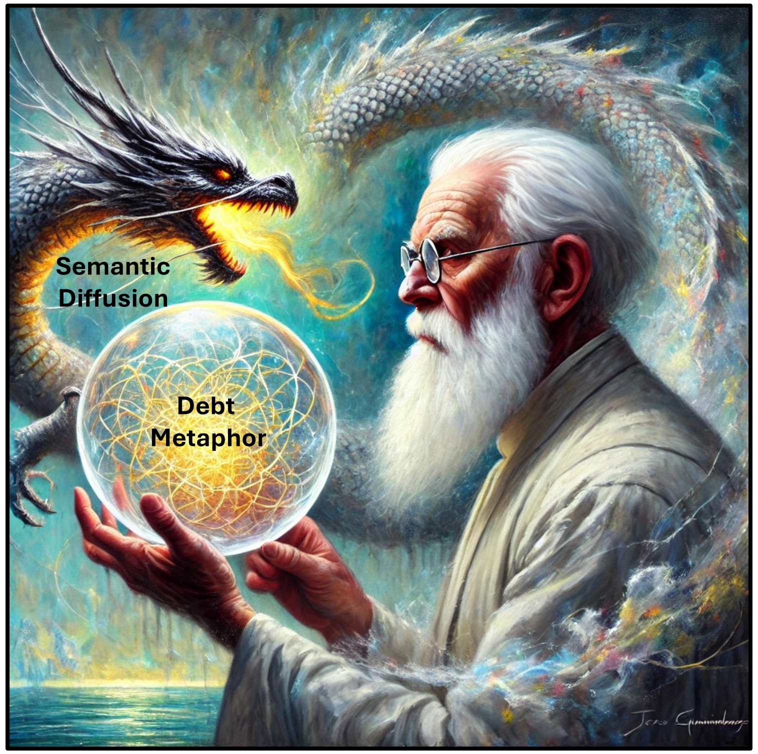
The Debt Metaphor
Ward Cunningham
in his 2009 YouTube video



slides by @philip_schwarz FP Iλλuminated https://fpilluminated.com/

Like many other terms in software development, the **Debt Metaphor** is not immune from what **Martin Fowler** calls **Semantic Diffusion**

@philip_schwarz

Semantic Diffusion

I have the habit of creating **neologisms** to describe the things I see in software development.

It's a common habit amongst writers in this field, for software development still lacks much useful jargon.

One of the problems with building a jargon is that **terms are vulnerable to losing their meaning, in a process of semantic diffusion** - to use yet another potential addition to our jargon.

Semantic diffusion occurs when you have a word that is coined by a person or group, often with a pretty good definition, but then gets spread through the wider community in a way that weakens that definition.

This weakening risks losing the definition entirely - and with it any usefulness to the term.

…

Martin Fowler
@martinfowler

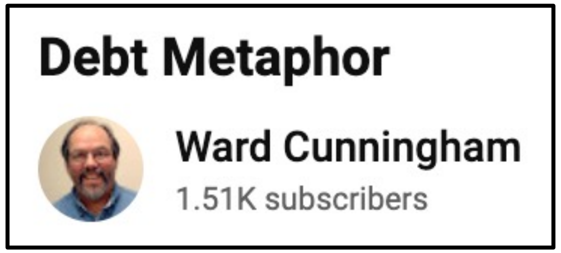
https://martinfowler.com/bliki/SemanticDiffusion.html

@philip_schwarz

This deck begins with a transcript of the **YouTube** video in which **Ward Cunningham**
- defines the **Debt Metaphor** (a term he coined)
- addresses the **confusion** he has noticed in some people's understanding of the term

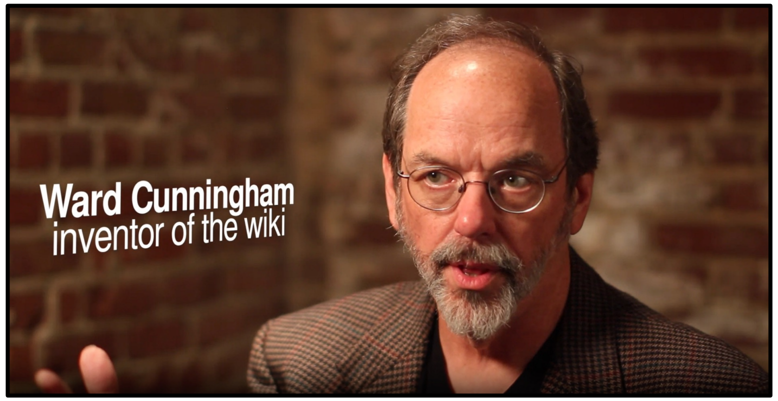
The deck continues with a **visual summary** of the video. The aim of the summary is twofold:
- provide a quick and easy reminder of the **metaphor**'s **original definition**
- help combat the **semantic diffusion** of the **metaphor**

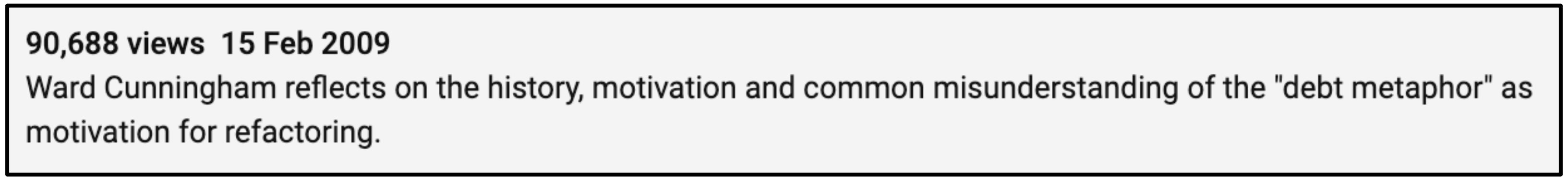
YouTube



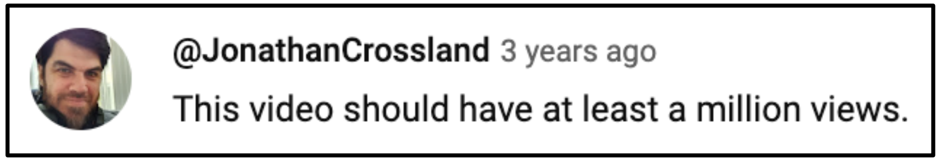
Debt Metaphor

Ward Cunningham
1.51K subscribers





90,688 views 15 Feb 2009
Ward Cunningham reflects on the history, motivation and common misunderstanding of the "debt metaphor" as motivation for refactoring.



@JonathanCrossland 3 years ago
This video should have at least a million views.

Ward Cunningham

Debt Metaphor

15 Feb 2009

Metaphor

I became interested in the way metaphors influence how we think, after reading George Lakoff and Mark Johnson's Metaphors We Live By.

An important idea is that we reason by analogy with the metaphors that have entered our language.

Debt

I coined the Debt Metaphor to explain the refactoring that we were doing on the WyCash product.

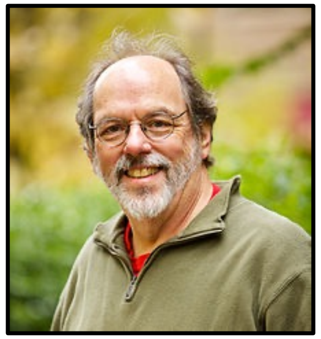
This was an early product done in Digitalk Smalltalk, and it was important to me that we accumulate the learnings we did about the application over time by modifying the program to look as if we had known what we are doing all along, and to look as if it had been easy to do in Smalltalk.

The explanation I gave to my boss, and this was financial software, was a financial analogy I called the debt metaphor, and that said that if we fail to make our program align with what we then understood to be the proper way to think about our financial objects, then we were going to continually stumble over that disagreement, and that would slow us down, which is like paying interest on a loan.

Speed

With borrowed money you can do something sooner than you might otherwise, but then, until you pay back that money, you'll be paying interest.

I thought borrowing money was a good idea, I thought that rushing software out the door to get some experience with it was a good idea, but that of course, you would eventually go back and as you learn things about that software, you would repay that loan by refactoring the program to reflect your experience as you acquired it.



Burden

I think that there were plenty of cases were people would rush software out the door and learn things but never put that learning back into the program, and that by analogy was borrowing money thinking that you never had to pay it back.

Of course if you do that, say with your credit card, eventually all your income goes to interest and you purchasing power goes to zero.

By the same token, if you develop a program for a long period of time by only adding features and never reorganizing it to reflect your understanding of those features, then eventually that program simply does not contain any understanding and all efforts to work on it take longer and longer, in other words, the interest is total, you'll make zero progress.

Agility

A lot of bloggers at least have explained the debt metaphor and confused it, I think, with the idea that you could write code poorly with the intention of doing a good job later and thinking that that was the primary source of debt.

I am never in favour of writing code poorly, but I am in favour of writing code to reflect your current understanding of a problem even if that understanding is partial.

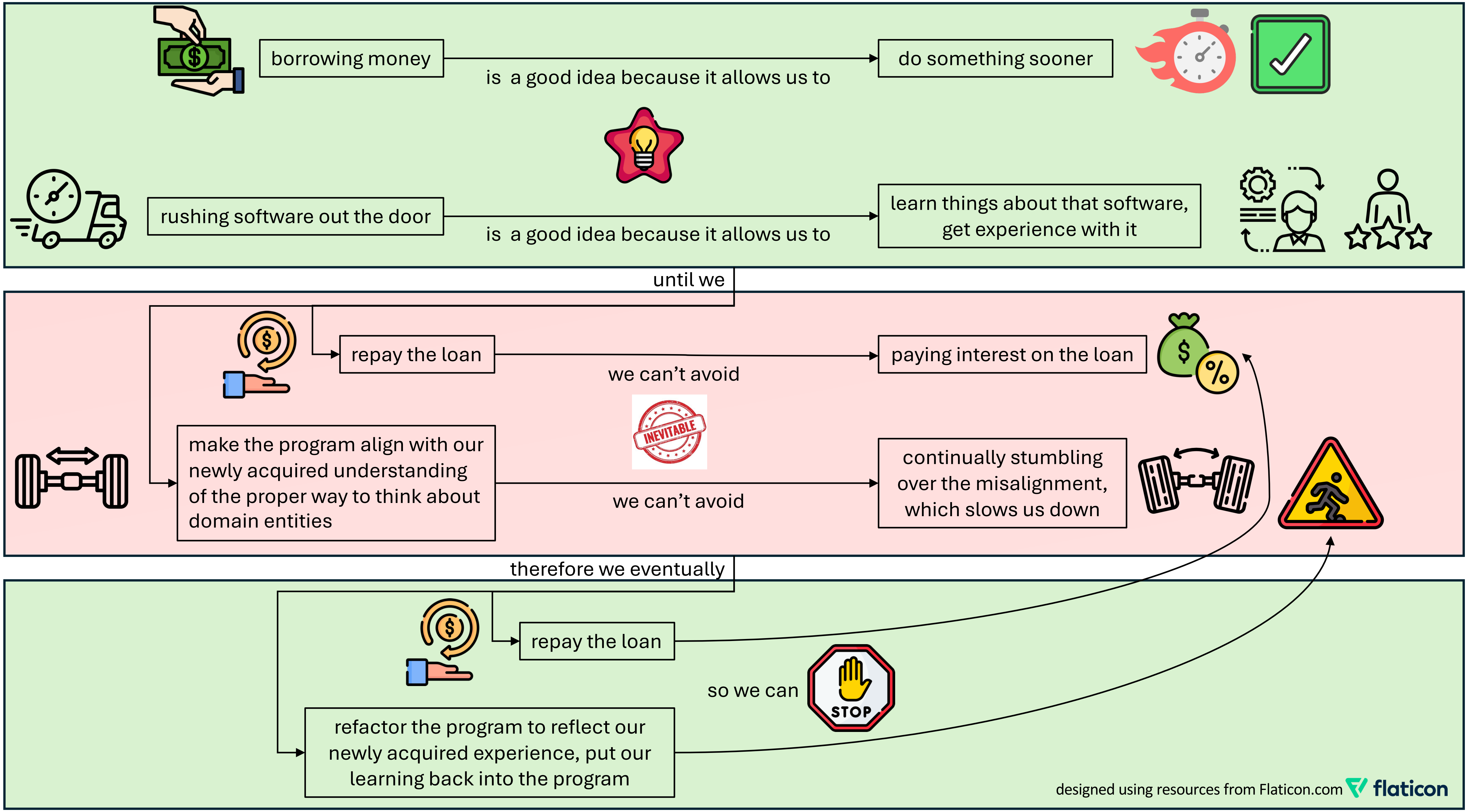
If you want to be able to go into debt that way, by developing software that you don't completely understand, you are wise to make that software reflect your understanding as best you can, so that when it does come time to refactor, it's clear what you were thinking when you wrote it, and making it easier to refactor it into what your current thinking is now.

In other words, the whole debt metaphor, or let's say, the ability to pay back debt, and make the debt metaphor work for your advantage, depends upon you writing code that is clean enough to be able to refactor as you come to understand your problem.

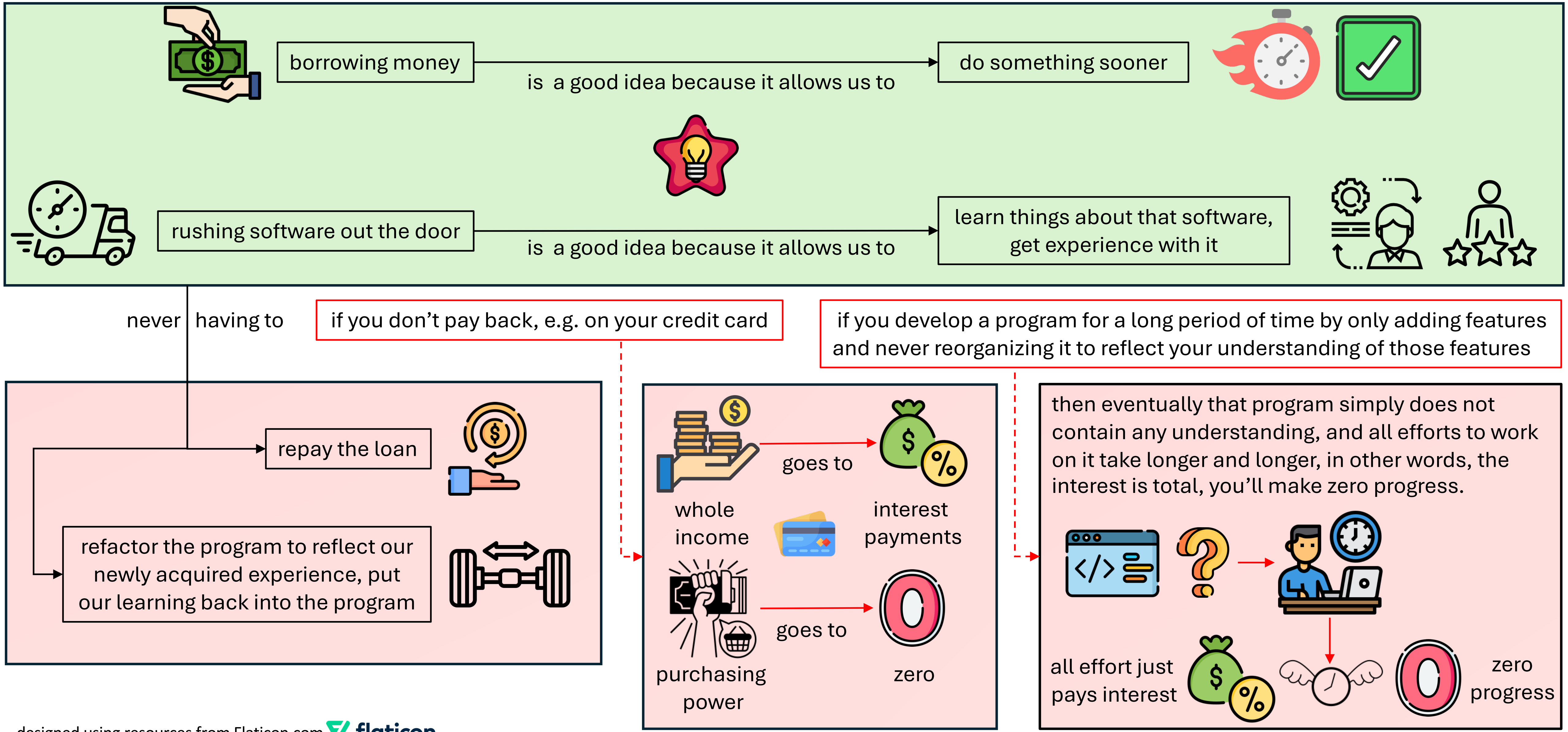
I think that's a good methodology, it is at the heart of extreme programming (XP).

The debt metaphor is an explanation, one of many explanations why XP works.

The remaining five slides are a visual summary of the video.


borrowing money
is a good idea because it allows us to
do something sooner
rushing software out the door
is a good idea because it allows us to
learn things about that software, get experience with it
until we
repay the loan
we can't avoid
paying interest on the loan
INEVITABLE
make the program align with our newly acquired understanding of the proper way to think about domain entities
we can't avoid
continually stumbling over the misalignment, which slows us down
therefore we eventually
repay the loan
so we can
STOP
refactor the program to reflect our newly acquired experience, put our learning back into the program
designed using resources from Flaticon.com
flaticon

Debt Metaphor
Confusion



borrowing money → is a good idea because it allows us to → do something sooner

rushing software out the door → is a good idea because it allows us to → learn things about that software, get experience with it

never | having to

repay the loan

refactor the program to reflect our newly acquired experience, put our learning back into the program

if you don't pay back, e.g. on your credit card

whole income → goes to → interest payments

purchasing power → goes to → zero

if you develop a program for a long period of time by only adding features and never reorganizing it to reflect your understanding of those features

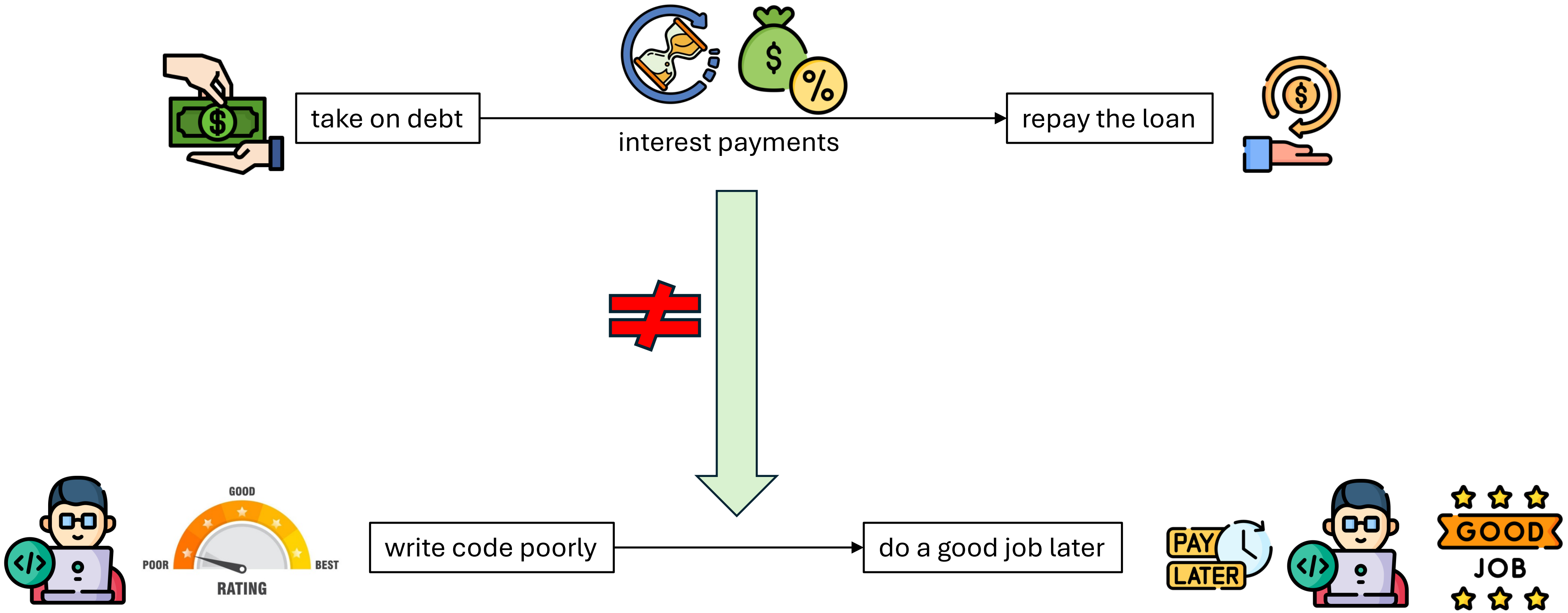
then eventually that program simply does not contain any understanding, and all efforts to work on it take longer and longer, in other words, the interest is total, you'll make zero progress.

all effort just pays interest

zero progress

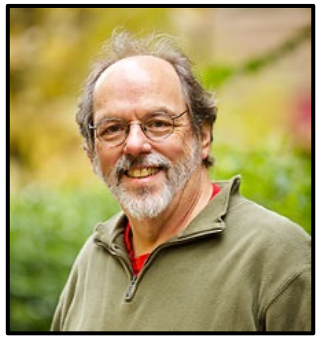
designed using resources from Flaticon.com flaticon

Debt Metaphor Confusion



"I am never in favour of writing code poorly"

designed using resources from Flaticon.com flaticon



I am **never** in favour of **writing code poorly**.
I **am** in favour of **writing code to reflect your current understanding of a problem** even if that **understanding** is **partial**

If you want to be able to go into **debt** that way…

by developing software that **you don't completely understand**

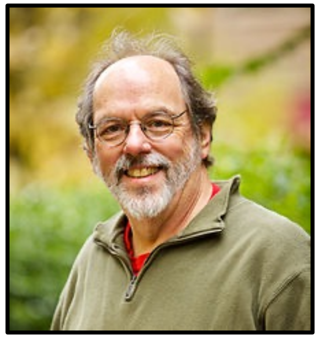
you are **wise** to

make that software **reflect** your **understanding** as best you can

so that when it does come time to **refactor**

It is **clear** what you were **thinking** when you wrote it

it is **easier** to **refactor** it into what your **current thinking** is now



The ability to pay back **debt**

and make the **debt metaphor** work for your advantage

Debt Metaphor

WORK for your ADVANTAGE

depends upon

you writing code that is **clean enough**

to be able to **refactor**

as you come to **understand** your **problem**

designed using resources from Flaticon.com flaticon

That’s all. I hope you found it useful.