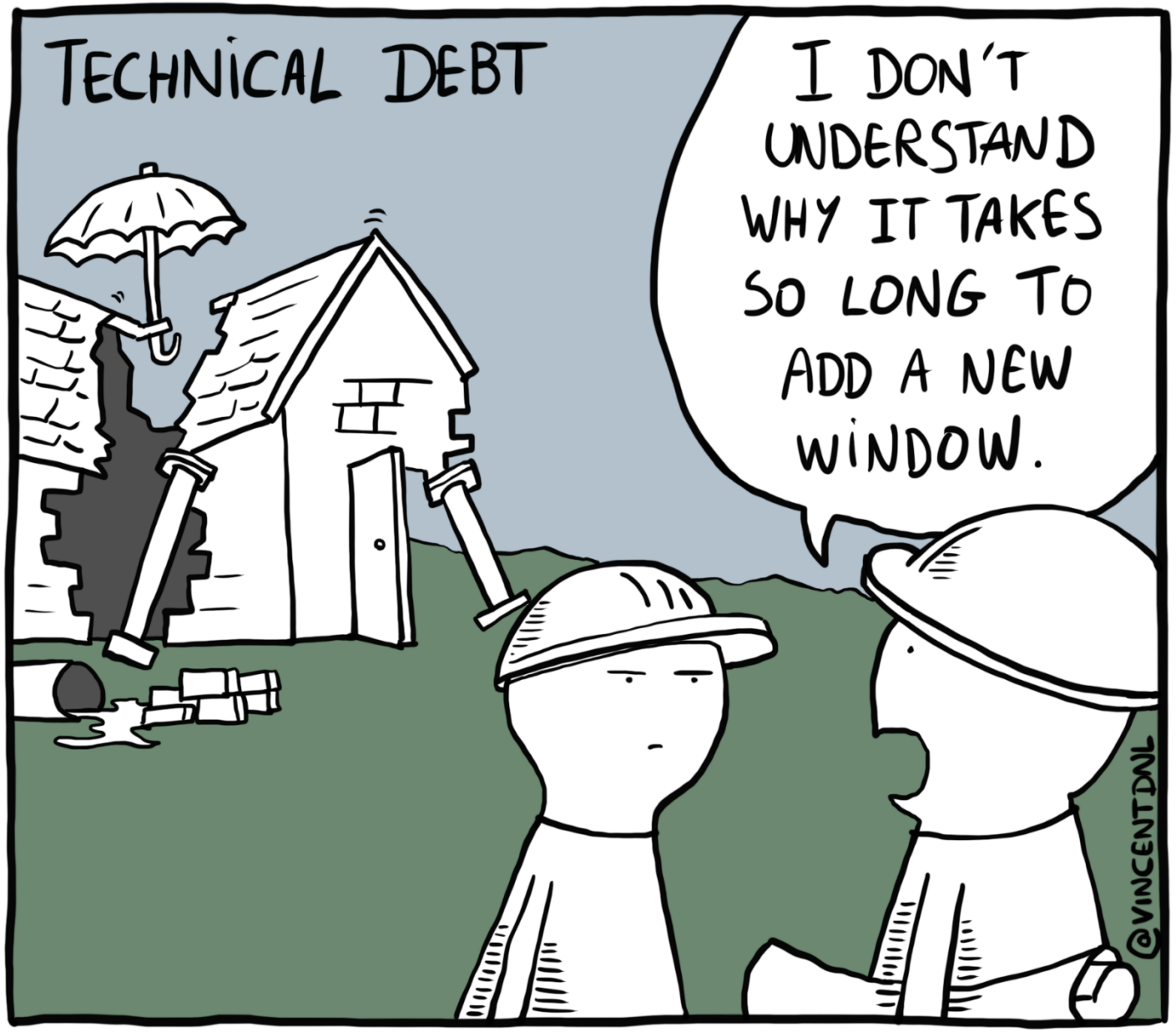
TECHNICAL DEBT
I DON'T UNDERSTAND WHY IT TAKES SO LONG TO ADD A NEW WINDOW.
@VINCENTDNL

DESIGN SMELLS

The oders of Rotting software

AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SWD
02 SEPTEMBER 2023

ALEXANDER MØLSTED HULGAARD RASMUSSEN
TEACHING ASSISTANT / IT SOFTWARE ENGINEER

THE WHAT, WHY AND HOW OF SOFTWARE ROT

- *What*: Inability to continuously support product
- *Why*: Design does not support (unexpected) changes in requirements
- *How*: Unexpected changes are hacked into existing code

https://www.youtube.com/watch?v=AbSehcT19u0

RIGIDITY AND FRAGILITY

- Rigidity
  - **Symptom**: apparently simple changes causes a cascade of changes in related component
  - **Problem**: Managers and/or programmers don't dare to make non-critical changes

I just needed to change A, but there were some expectations to A hard-coded in B when it used C and D, so to change that I had to change B, C and D, but then…

- Fragility
  - **Symptom**: Fixing one problem introduces more problems
  - **Problems**: Software like this is impossible to maintain. Programmers cannot trust that errors isn't introduced.

Fixing A looked really simple. I didn't expect B to break, though, and when I tried to fix that I inadvertently broke C and D, too…

IMMOBILITY AND VISCOSITY

- Immobility
  - **Symptom**: A component could be reused in some other place, but depends on to much ‘bagage’, for it to be easily reused
  - **Problem**: Software components are not reused, but rewritten

The `Calendar` class really has generic functionality, but it relies on the DS1339A RTC, so we can’t reuse the software

- Viscosity
  - Given a problem – there are usually more then one way to solve it, some preserve the design, others do not i.e hack.
  - **Symptom**: It is easier to solve a problem by making a hack than preserve the design
  - **Problem**: The software degenerates – because to many hacks, workarounds, shortcuts etc. are introduced

It’s just sooo much quicker to put an `#ifdef` there than to redesign the component…yeah.

NEEDLESS COMPLEXITY & REPETITIONS

- Needless complexity (YAGNI)
  - When a software system is “overengineered” to account for possible future changes (there is a balance)

The `Report` class uses an XML file and is prepared for the use of a printer, a speech synthesizer, smoke signals, Morse code and telepathy. I’m sure my boss will be pleased!

- Needless repetitions (DRY)
  - Changes implemented by copy’n’paste instead of proper abstration

The code is right there, so why not just CnP it? I only need to change a little bit, and if it works for A it’ll work for B…and C, and D

OPACITY

- Code evolves to become harder and harder to understand in lieu of proper refactoring.

```
public bool IsPrinterOn() {
  return printer.IsOn();
}
```

```
public bool IsPrinterOn() {
  if(!printer.IsOn()) {
    printer.WakeFromSleepMode();
    while(i<125000) i++;
    return printer.IsOn();
  }
  return true;
}
```

```
public bool IsPrinterOn() {
  if(!(curPS = printer.IsOn()))
  {
    printer.WakeFromSleepMode();
    while(i<125000) i++;
    curPS = printer.IsOn();
    if(!curPS)
      goto CHECK_ON_AGAIN;
  }
  return true;

CHECK_ON_AGAIN:
  while(i<32500) i++;
  if(!printer.IsON())
    return false; //probably
}
```

BOTTOM LINE


Tweet
Programming Wisdom @CodeWisdom · Jan 21, 2018
“If you can get today’s work done today, but you do it in such a way that you can’t possibly get tomorrow’s work done tomorrow, then you lose.” - Martin Fowler, Refactoring: Improving the Design of Existing Code
10
640
1.3K
Replies
Martin Fowler @martinfowler · Jan 21, 2018
Replying to @CodeWisdom
quote is actually by @KentBeck . It appears in Refactoring, but in a sidebar that he wrote.
1
10
132

AARHUS
UNIVERSITY

REFERENCES

Technical Debt: www.larochelab.com