“There is no code so big, twisted, or complex that maintenance can't make it worse.”

- Gerald M. Weinberg
Software Maintenance

- Anti-Patterns:
  - Informal bug tracking
  - Not allocating post-release staffing
    - Bad prior release distracts team
  - Not paying off technical debt

- Code maintenance during and after development
  - You need a process to identify bugs and track to resolution
  - Most software is an update, not a clean-slate project
  - Ongoing effort is required to repay “technical debt”

https://goo.gl/Crc1zq
Managing Bugs

- Map reported issue to an actual bug
  - L1/L2/L3 support to capture bug report
  - Sorting out duplicate reports takes effort
- Prioritize the bug fix (e.g., risk table)
  - Combination of frequency, business cost
- Find someone with right skills to fix it
  - Does this derail new development tasks?
  - Quick and dirty? Or a solid re-engineer fix?
- Validate the fix
  - Did you inject a new fault with the fix?
- Package the fix and deploy it
  - Hot patch? Defer to future schedule release?

Risk table example:
- High consequence defect
- With low probability of occurrence
  ➔ Medium risk / medium priority bug
Most SW work is on existing code, not a clean slate
  - “Clean slate” often works with COTS components

60/60 rule [Glass, *IEEE Software* May 2001]
  - Maintenance can average 60% of lifecycle cost
  - About 60% of maintenance is adding new features

Maintenance is harder than development
  - Need to understand existing system
    - Motivation for keeping entire V document chain up to date
    - Optimized code is more painful to maintain
  - Need to modify system without breaking things
    - Complete rewrite usually impractical – and might be worse

https://goo.gl/1CqN9i
Managing Technical Debt

- Technical debt: messy code/design/architecture that hasn’t been cleaned up
  - Some signs of debt:
    - Degraded code quality (spaghetti code, globals, warnings, …)
    - Skipped process steps (missing peer reviews, unit tests, …)
    - High fault reinjection ratio (new bugs when fixing old bugs)
  - You incur debt by taking a shortcut
    - Short-term debt can be useful (e.g., meet a deadline)
  - Repay debt by refactoring the system

- Technical debt incurs interest
  - Shortcuts often lead to bugs, fragility
  - Accumulated debt becomes unsustainable

- Use the right amount of debt
  - It’s like using a credit card responsibly
  - Devote part of each development cycle to repaying technical debt

https://goo.gl/cFXrD9
Best Practices for Maintenance

- Most development is maintenance
  - Plan for and staff maintenance
    - Most development is on the next revision
    - Plan for high priority emergency fixes
  - Keep up with technical debt payments

- Maintenance pitfalls
  - Not allocating time for bugs, maintenance & technical debt
    - For example, need perhaps 10% budget for technical debt repayment
    - Leave slack in deadlines for fixing urgent previous-version bugs
  - Evaluating programmers only for clean-sheet development skills

https://goo.gl/DDZfCY
Just put the technical debt on my credit card

Moving Fast and Breaking Things
Fragile Development Guide

O RLY?
@ThePracticalDev
CHANGES IN VERSION 10.17:
The CPU no longer overheats when you hold down spacebar.

COMMENTS:

LONGTIME_USER4 writes:
This update broke my workflow! My control key is hard to reach, so I hold spacebar instead, and I configured Emacs to interpret a rapid temperature rise as "control".

ADMIN writes:
That’s horrifying.

LONGTIME_USER4 writes:
Look, my setup works for me. Just add an option to reenable spacebar heating.

EVERY CHANGE BREAKS SOMEONE’S WORKFLOW.

https://xkcd.com/1172/