Key Development Metrics

"If you torture the data long enough, it will confess to anything."

– *Ronald Coase*

Key Embedded Software Metrics

- Anti-Patterns:
  - Development effort > validation effort
  - Too many lines of code per hour
  - Peer review finds <50% of all bugs

- Healthy project metrics:
  - About 2-3 hours of validation effort per hour development
    - Tester:Developer head count ratio is about 1 to 1
  - Productivity of 1-2 lines of code per hour for solid software
    - This includes entire process (requirements through acceptance test)
  - Peer review should be finding >50% of all defects

Software = Design + Testing

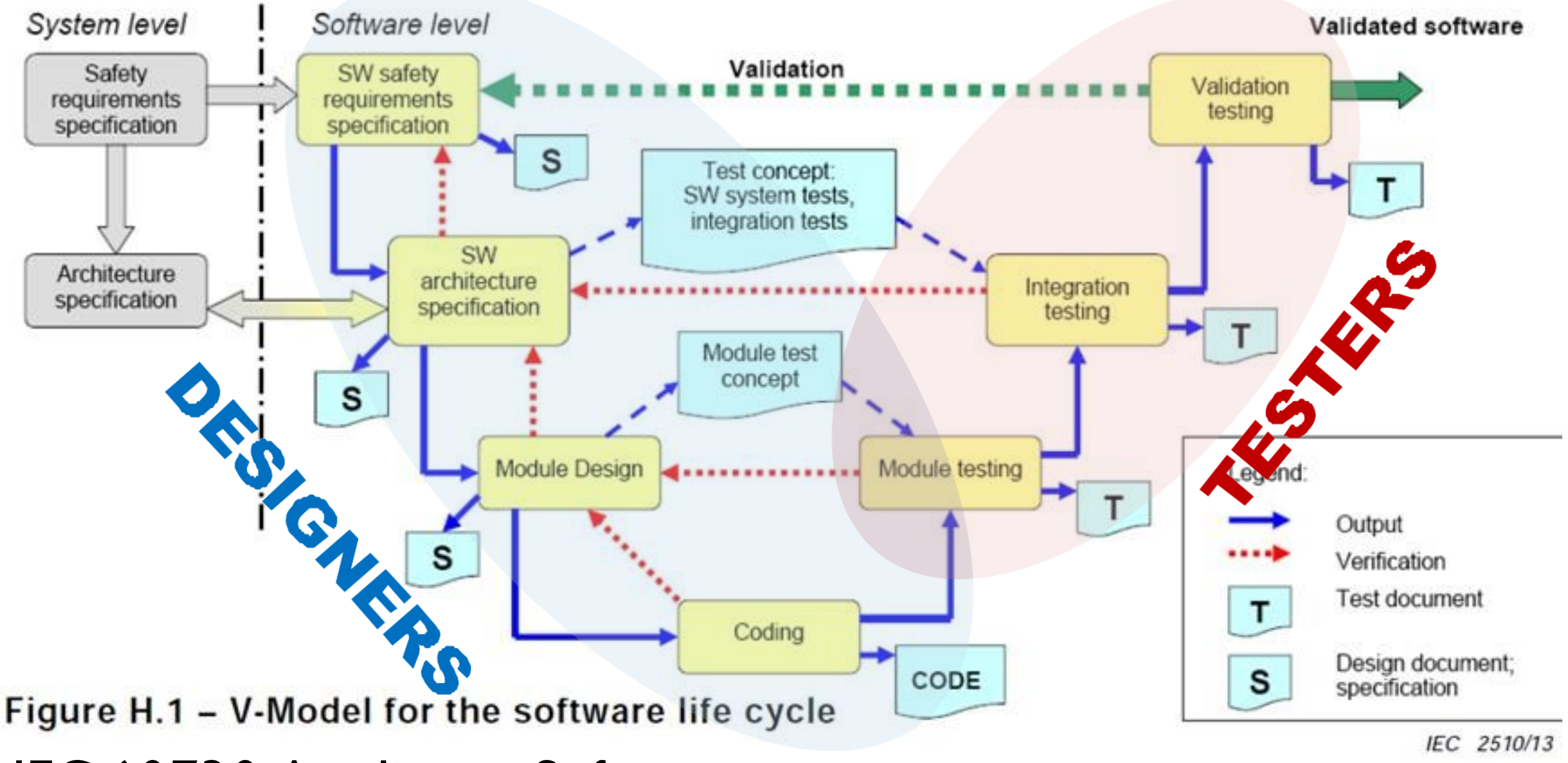


Figure H.1 – V-Model for the software life cycle

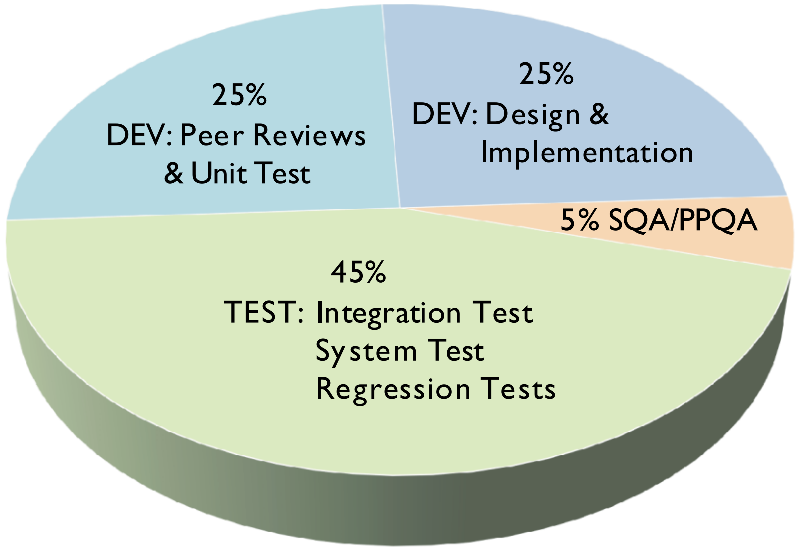
IEC 60730 Appliance Safety

[IEC 60730]

Typical Effort Distribution

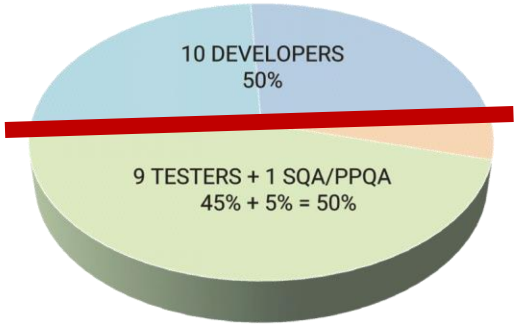
- Tester to Developer ratio varies depending on situation
  - Web development: 1 tester per 5-10 developers
  - Microsoft: 1 tester per 1 developer
  - Aircraft controls: ~5 testers per 1 developer

EMBEDDED SW PROJECT EFFORT

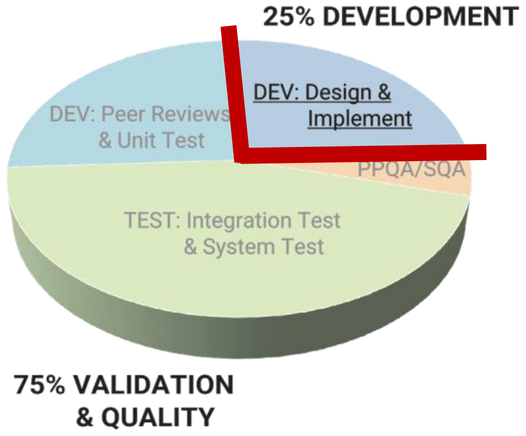


50%/50% Head Count

20 Person Project



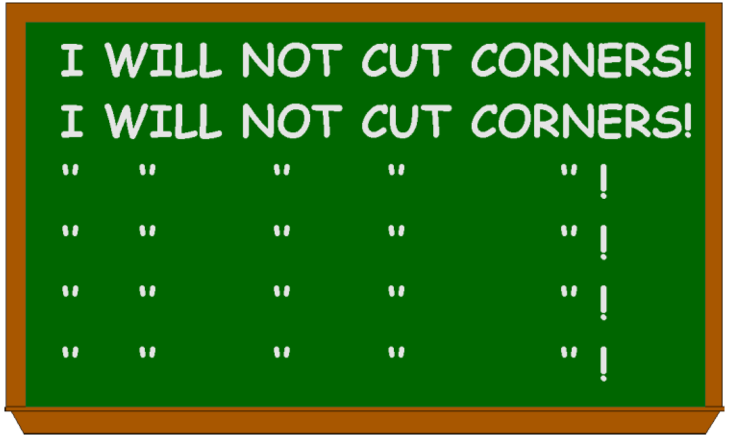
25%/75% Effort



Code Productivity

- Productivity 1-2 lines of code/hr (including testers)
  - Perhaps 3 lines/hr with Agile, but that speed increases quality risk

- High lines of code/hr ➔ cutting corners
  - Partial requirements, no design?
  - No peer reviews?
  - Only system level testing?



[Simpsons 7F11]

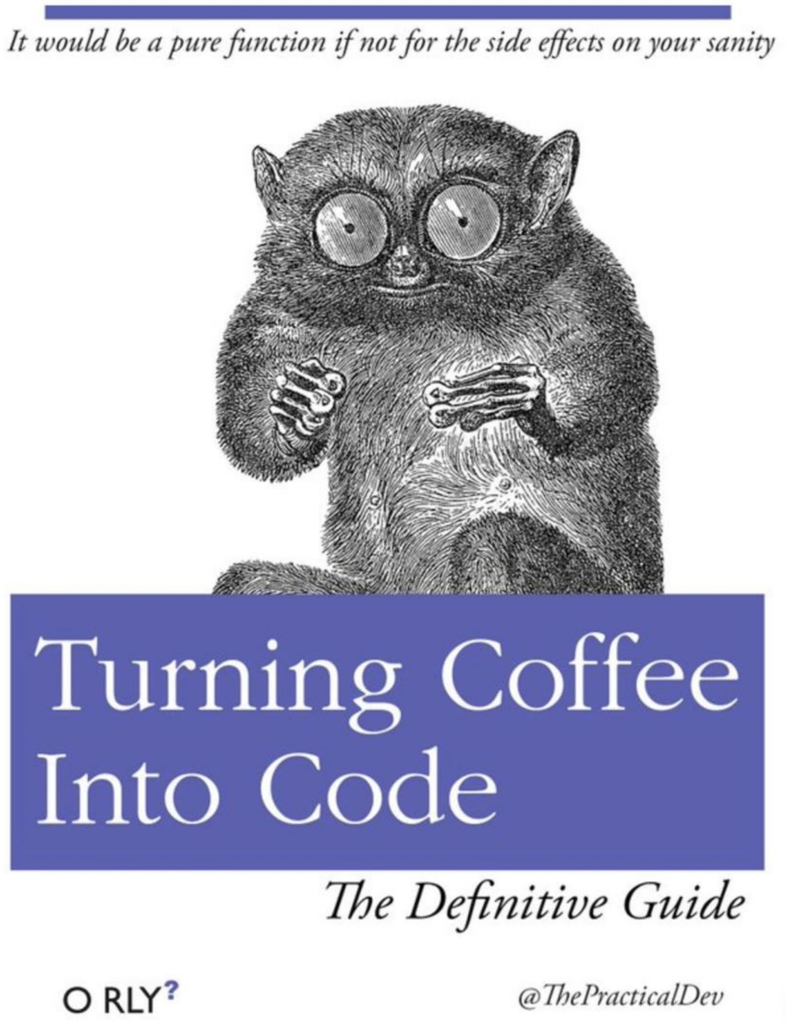
- $25-$75 / line of source code
  - All-in cost, including entire V process, until field testing
  - “Maintenance” can cost more, but might count as new project

Peer Review Effectiveness

- Good peer reviews find 50%-70% of the defects
  - Fewer than 40%-50% of defects found in peer reviews mean they are BROKEN
- Peer Reviews cost perhaps 5%-10% of total project cost
  - Let's do the math:
    - Peer reviews process about 100 lines of code per hour total
    - Three reviewers ➔ 33 lines of code per person-hr
      = 0.033 hours per line of code reviewed (2 minutes)
    - 0.033 hours review / .5 hours per LOC total = 6.7% for code review
    - Plus review requirements & design …but still a great ROI
- Are peer reviews finding half your bugs?
  - Are you spreading them out or bunching them together?
  - If they're not finding bugs, consider improving review culture

Best Practices For Key Software Metrics

- 2-3 hours of validation for each 1 hour of development
  - Head count ratio generally 1 Tester to 1 Developer
  - About 5% of effort for SQA
- Code productivity of about 1 to 3 lines per hour
  - At or above 3 lines/hr, you probably are cutting corners
- Peer reviews should find 50% (or more) of defects
  - At about 5%-10% of total project cost
- Metric Pitfalls
  - Use only metrics that provide value – don't go crazy with metrics!
  - Gaming the metric doesn't improve software quality
  - Reward/punish based on metric values will render metric useless



It would be a pure function if not for the side effects on your sanity

Turning Coffee Into Code

The Definitive Guide

O RLY?

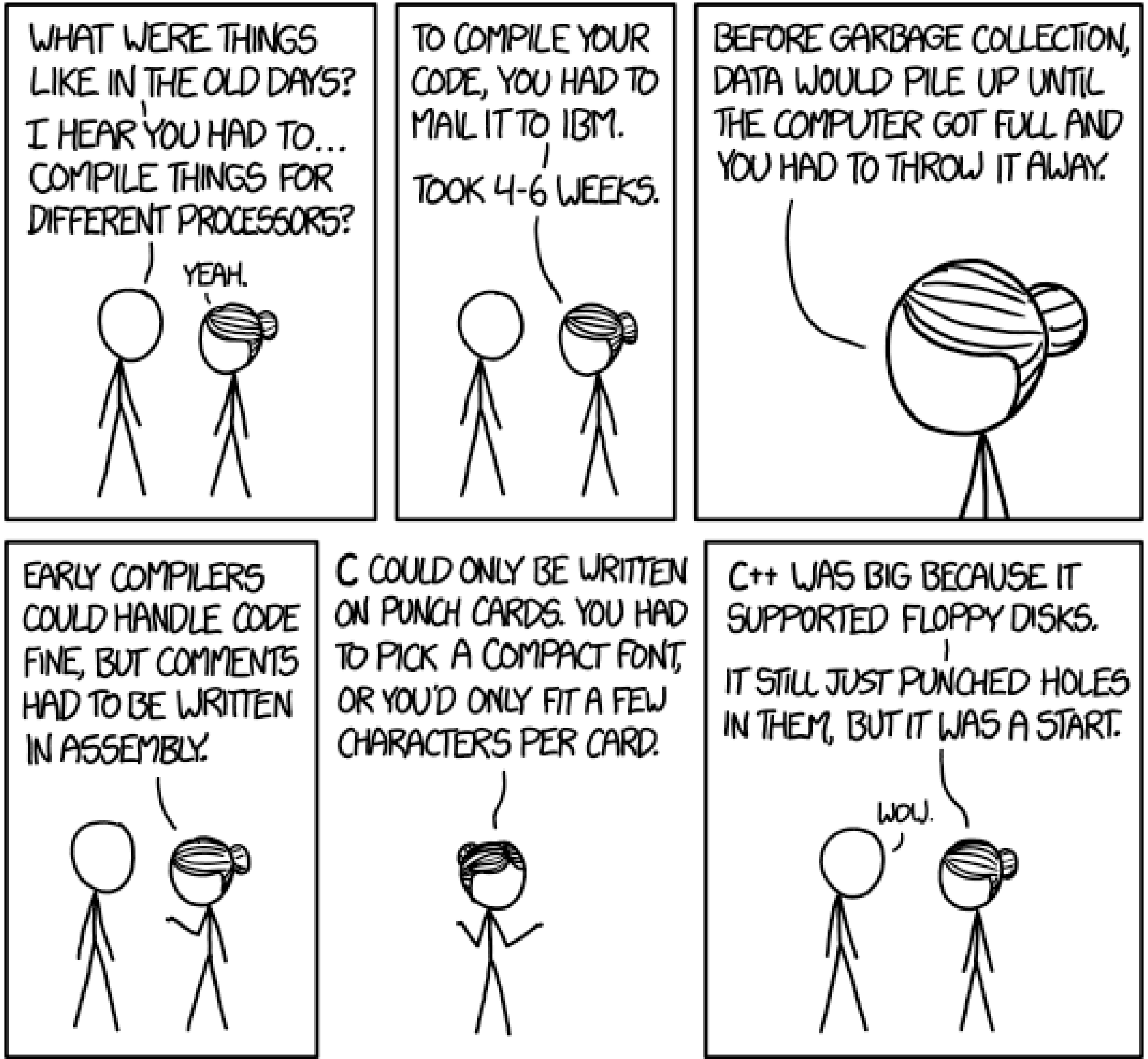
@ThePracticalDev

GOOD

FAST

CHEAP

(Pick Any Two)


WHAT WERE THINGS LIKE IN THE OLD DAYS?
I HEAR YOU HAD TO... COMPILE THINGS FOR DIFFERENT PROCESSORS?
YEAH.
TO COMPILE YOUR CODE, YOU HAD TO MAIL IT TO IBM.
TOOK 4-6 WEEKS.
BEFORE GARBAGE COLLECTION, DATA WOULD PILE UP UNTIL THE COMPUTER GOT FULL AND YOU HAD TO THROW IT AWAY.
EARLY COMPILERS COULD HANDLE CODE FINE, BUT COMMENTS HAD TO BE WRITTEN IN ASSEMBLY.
C COULD ONLY BE WRITTEN ON PUNCH CARDS. YOU HAD TO PICK A COMPACT FONT, OR YOU'D ONLY FIT A FEW CHARACTERS PER CARD.
C++ WAS BIG BECAUSE IT SUPPORTED FLOPPY DISKS.
IT STILL JUST PUNCHED HOLES IN THEM, BUT IT WAS A START.
WOW.

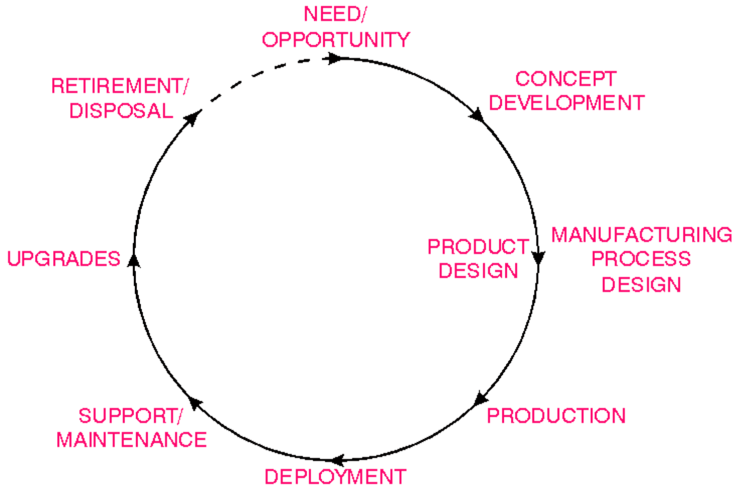
https://m.xkcd.com/1755/

Lifecycle & Configuration Management

"Good judgment comes from experience,
and experience comes from bad judgment."
– *Frederick P. Brooks*

Lifecycle Issues

- Anti-Patterns:
  - No version control
  - No configuration management
  - Incremental features without baseline



- Lifecycle issues
  - Version control: keeping different versions straight
  - Configuration management: what's in the deployed system
  - Lifecycle: old embedded systems (almost) never die
    - Spare parts for obsolete systems
    - Mid-life upgrades

INSTALLATION ERROR —

Airbus confirms software configuration error caused plane crash

Airbus A400M flight recorder data confirms "quality issue" in setup caused failure.

SEAN GALLAGHER - 6/1/2015, 12:38 PM

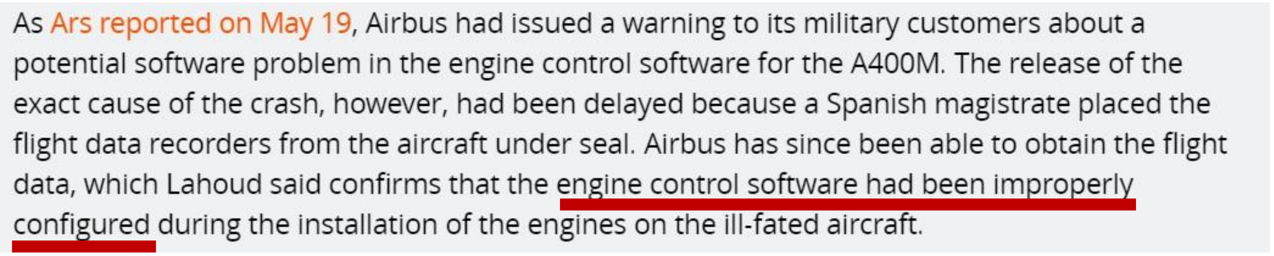
A protoype of the Airbus A400M at the 2010 Farnborough Airshow.

https://www.youtube.com/watch?v=TUiX6m6WLdY&ab_channel=Sciences

The Register 6/10/15:

"Torque calibration parameters were wiped"…

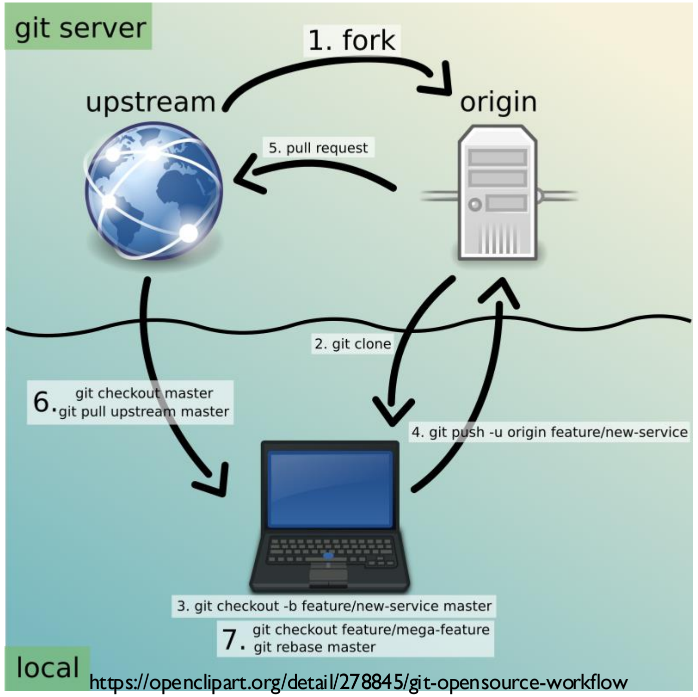
"Pilots only get warning above 120 meters off the ground"


As Ars reported on May 19, Airbus had issued a warning to its military customers about a potential software problem in the engine control software for the A400M. The release of the exact cause of the crash, however, had been delayed because a Spanish magistrate placed the flight data recorders from the aircraft under seal. Airbus has since been able to obtain the flight data, which Lahoud said confirms that the engine control software had been improperly configured during the installation of the engines on the ill-fated aircraft.

http://arstechnica.com/information-technology/2015/06/airbus-confirms-software-configuration-error-caused-plane-crash/

Version Control

- Stores & navigates snapshots of versions
  - Ability to roll-back to previous version
  - Synchronizes compatible versions
    - Development vs. release versions
    - "Branch" and "merge" parallel efforts
  - Given a version number, give me the software
- Many popular tools – use one!
  - Watch out for binary file management
  - Multi-site can require special care
- Beyond the obvious…
  - Also version: design documents, requirements, tests, tool chain
  - Need process for who/what/when to update version
  - Needs to tie into disaster recovery (have you tested it?)

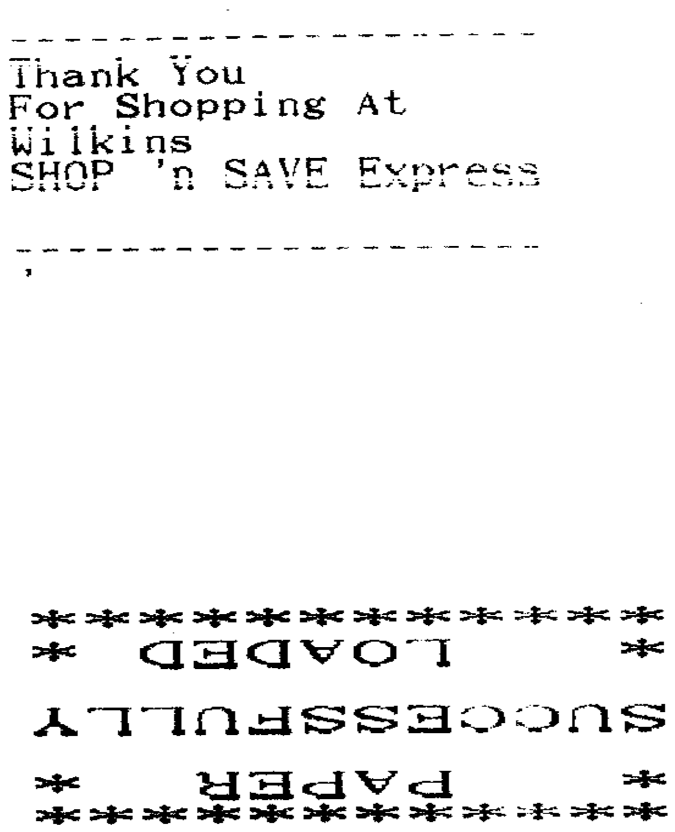

git server
1. fork
upstream
origin
5. pull request
2. git clone
6. git checkout master git pull upstream master
4. git push -u origin feature/new-service
3. git checkout -b feature/new-service master
7. git checkout feature/mega-feature git rebase master
local
https://openclipart.org/detail/278845/git-opensource-workflow

Configuration Management

- CM is identifying a particular version
  - Which version has which bug fixes?
  - Which version has which features?
  - Which version should be the next release?
  - Which version is in a failed product?
- Beyond the obvious…
  - Snapshot every build and record its manifest
  - Make sure returned units can self-report version
    - Which library version, which driver version, etc.
    - Need SW version, HW version, config data version (if applicable)
    - Attempts to track this in a central database never work 100%
  - Need to know which SW is compatible with which HW
    - Need to be able to re-create a build, which might require obsolete tools
  - Feature activation: which features have been licensed by user?

Airbus A-380 bolt with part tracking information. (Size: 2 cm x 1 cm)

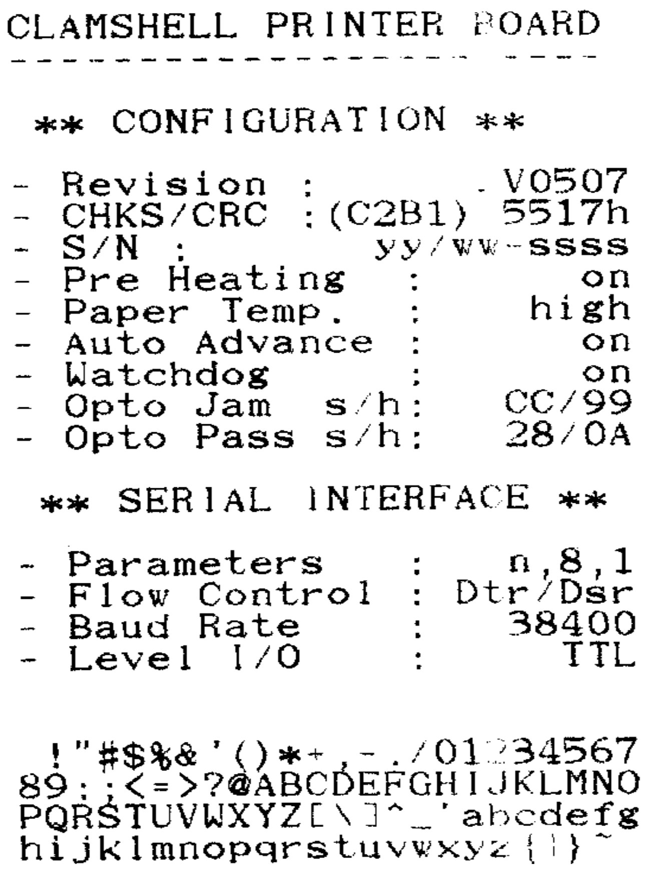
Gas Pump Startup Printout Example



```
------------------------
Thank You
For Shopping At
Wilkins
SHOP 'n SAVE Express

------------------------
```

```
************
*  PAPER   *
SUCCESSFULLY
*  LOADED  *
************
```



```
CLAMSHELL PRINTER BOARD
-----------------------

 ** CONFIGURATION **

- Revision     :        V0507
- CHKS/CRC    :(C2B1) 5517h
- S/N  :          yy/ww-ssss
- Pre Heating      :       on
- Paper Temp.      :     high
- Auto Advance     :       on
- Watchdog         :       on
- Opto Jam   s/h:       CC/99
- Opto Pass  s/h:       28/0A

 ** SERIAL INTERFACE **

- Parameters       :    n,8,1
- Flow Control     :  Dtr/Dsr
- Baud Rate        :    38400
- Level I/O        :      TTL

 !"#$%&'()*+,-./01234567
89:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_'abcdefg
hijklmnopqrstuvwxyz{|}~
```

Embedded Systems Live (almost) Forever

- 10-50 year lifecycles are common
  - Example: SAGE air defense system
    - Started 1954; deployed 1963
    - Vacuum tubes; 500,000 lines of code
    - In operation until 1983
  - 1AESS telephone switch: 1976 through 2008+
  - Aircraft can operate for 25-30 years
  - Cars routinely operate for 15+ years

https://goo.gl/X4wUBT

- Challenges
  - Disaster recovery (includes vendor bankruptcy)
  - End-of-life for hardware & software
  - Adding new services to existing platform
  - Security problems if not updated

https://goo.gl/EgtvwX

Britain's Doomsday Nuke Subs Still Run Windows XP

The fate of the country's nukes is in the hands of an obsolete operating system.

By Kyle Mizokami Jan 21, 2016

https://www.popularmechanics.com/military/weapons/a19061/britains-doomsday-subs-run-windows-xp/

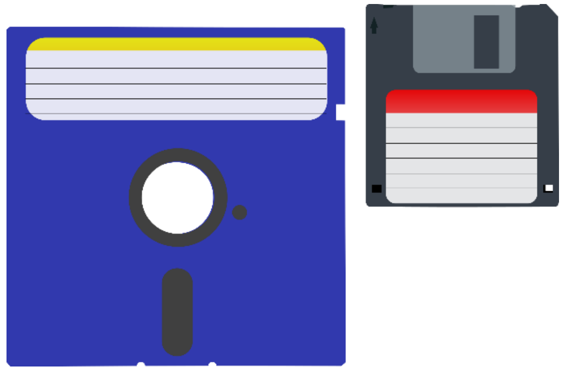
Posted on April 26, 2018 at 12:59 PM

WINDOWS 95 POWERED MEDICAL EQUIPMENT ARE BEING HIT BY HACKERS

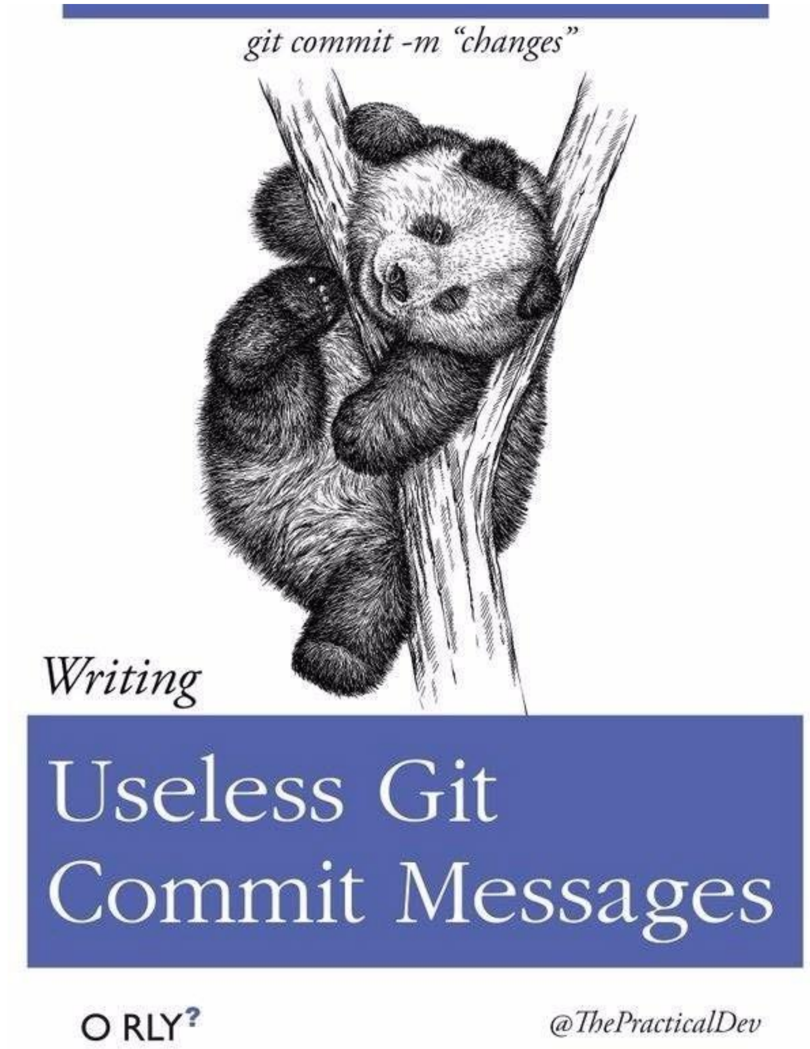
https://koddos.net/blog/windows-95-powered-medical- equipment-are-being-hit-by-hackers/

Best Practices for Lifecycle Management

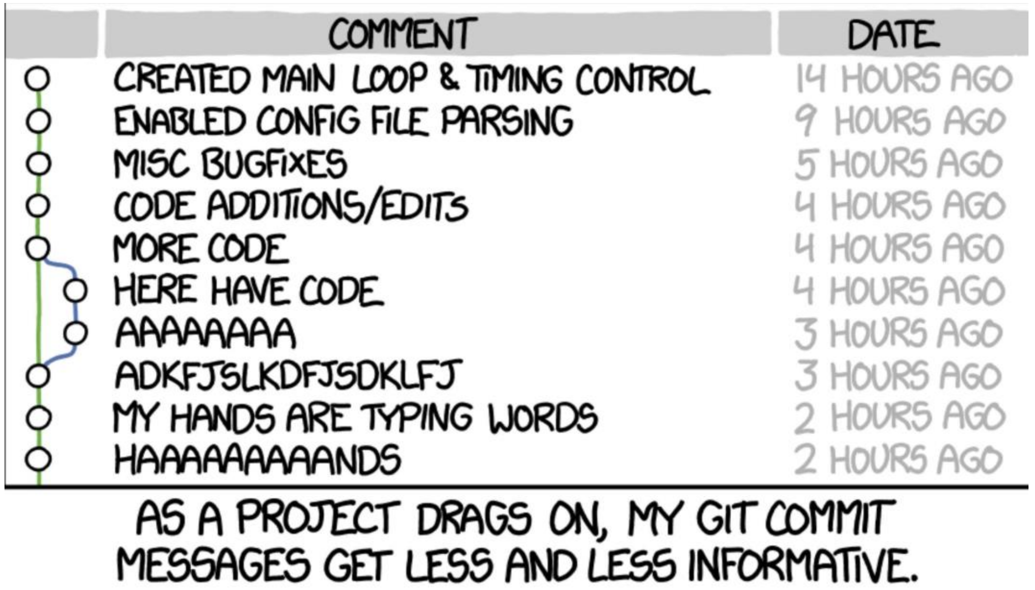
- Take CM & Versioning seriously – they are different
  - Use a formal Build Process to release
    - Check that all versions in build are correct; record manifest
  - Plan for 2x the lifecycle you think will happen
    - And have a plan for product end of life



- Lifecycle pitfalls
  - Releasing the wrong version
    - Development version, debug version (e.g., watchdog turned off), etc.
  - Cutting corners on software configuration management
    - What if you can't reconstruct software involved in a field failure?
  - Getting caught without replacement parts
    - Pay attention to end-of-life buys
    - Anticipate obsolete: hardware, media, tools, libraries, …



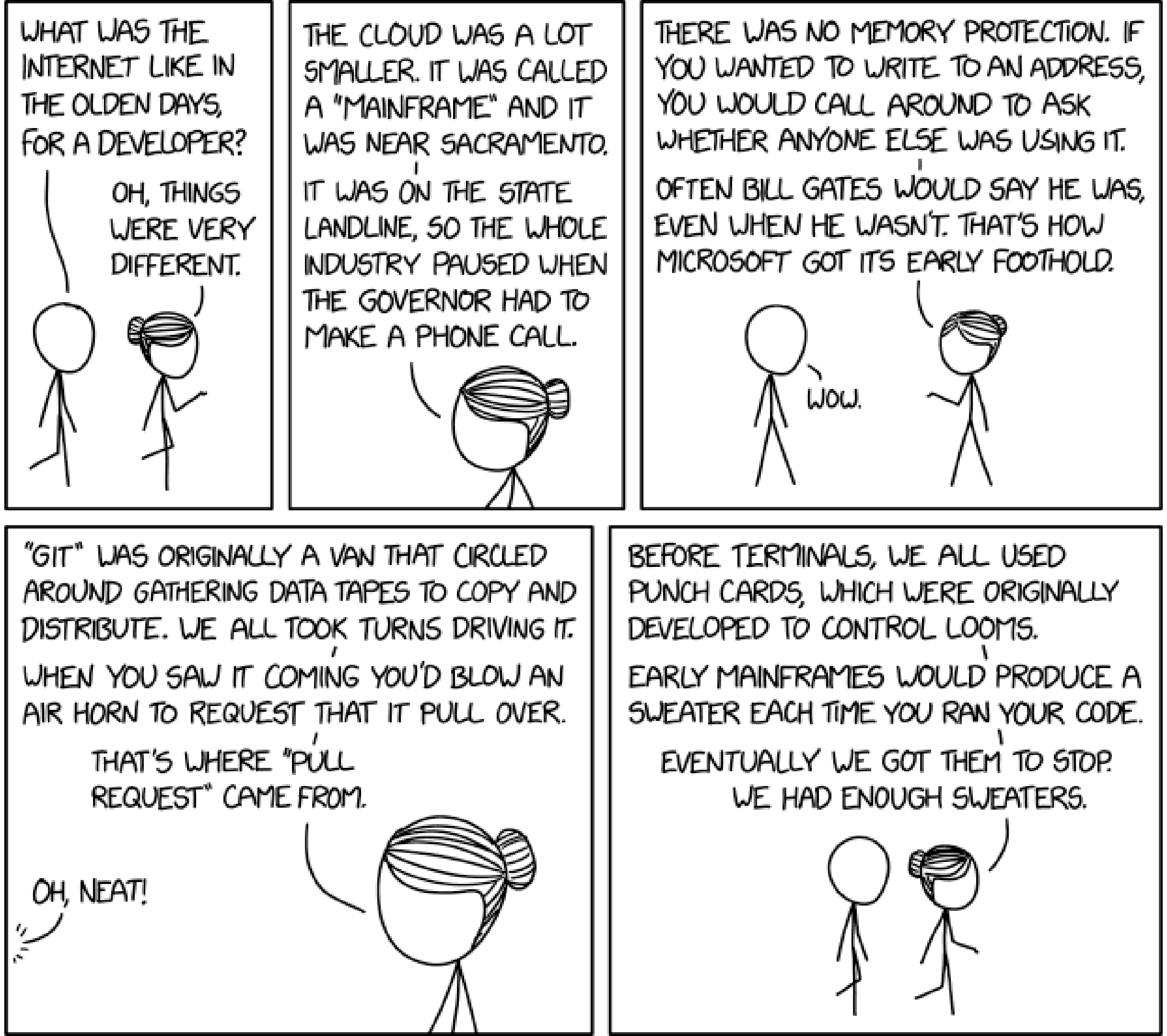
https://goo.gl/pvDMHX CC BY-NC 2.0



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

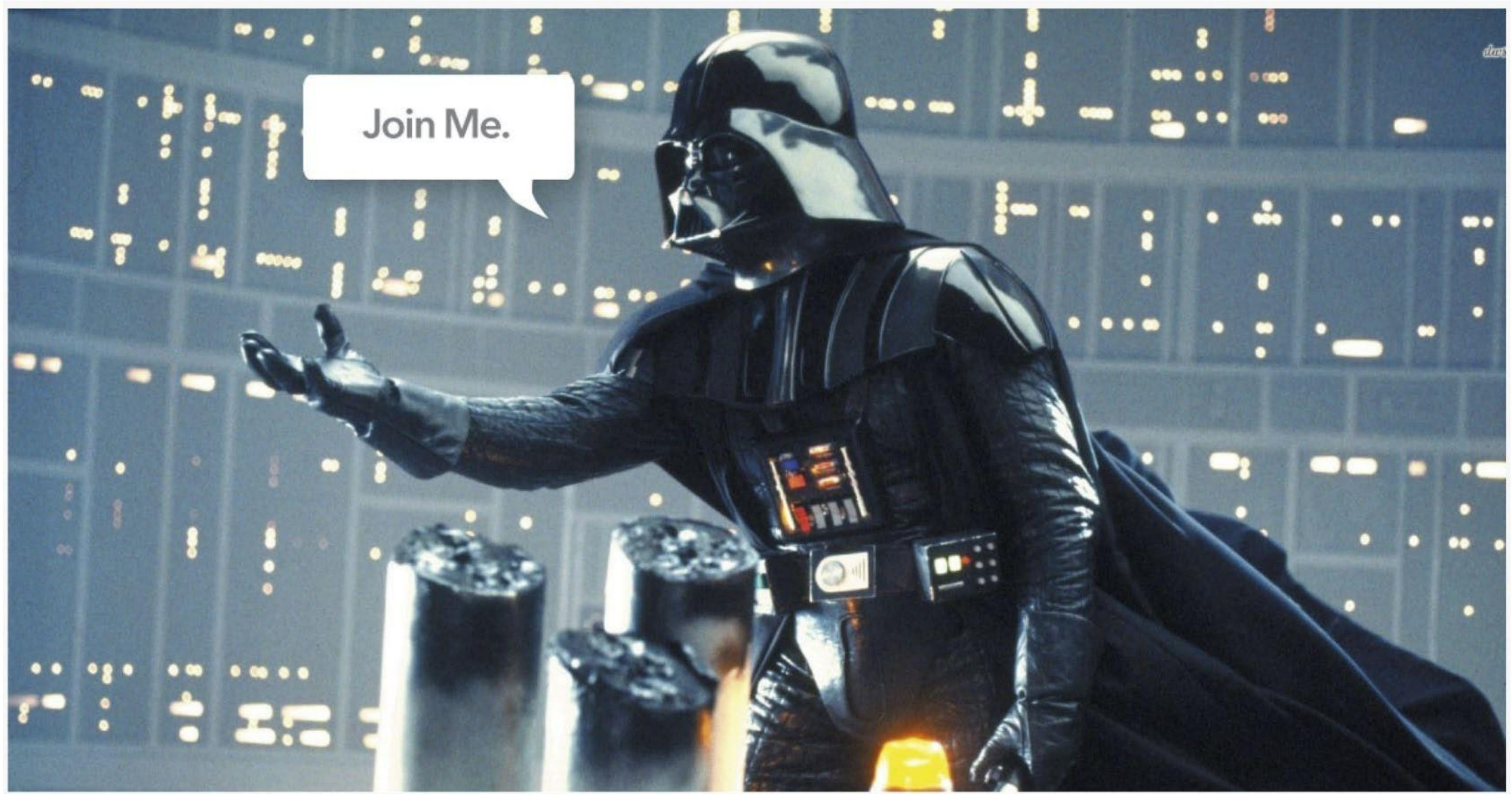
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

https://xkcd.com/1296/


WHAT WAS THE INTERNET LIKE IN THE OLDEN DAYS, FOR A DEVELOPER?
OH, THINGS WERE VERY DIFFERENT.
THE CLOUD WAS A LOT SMALLER. IT WAS CALLED A "MAINFRAME" AND IT WAS NEAR SACRAMENTO.
IT WAS ON THE STATE LANDLINE, SO THE WHOLE INDUSTRY PAUSED WHEN THE GOVERNOR HAD TO MAKE A PHONE CALL.
THERE WAS NO MEMORY PROTECTION. IF YOU WANTED TO WRITE TO AN ADDRESS, YOU WOULD CALL AROUND TO ASK WHETHER ANYONE ELSE WAS USING IT.
OFTEN BILL GATES WOULD SAY HE WAS, EVEN WHEN HE WASN'T. THAT'S HOW MICROSOFT GOT ITS EARLY FOOTHOLD.
WOW.
"GIT" WAS ORIGINALLY A VAN THAT CIRCLED AROUND GATHERING DATA TAPES TO COPY AND DISTRIBUTE. WE ALL TOOK TURNS DRIVING IT.
WHEN YOU SAW IT COMING YOU'D BLOW AN AIR HORN TO REQUEST THAT IT PULL OVER.
THAT'S WHERE "PULL REQUEST" CAME FROM.
OH, NEAT!
BEFORE TERMINALS, WE ALL USED PUNCH CARDS, WHICH WERE ORIGINALLY DEVELOPED TO CONTROL LOOMS.
EARLY MAINFRAMES WOULD PRODUCE A SWEATER EACH TIME YOU RAN YOUR CODE.
EVENTUALLY WE GOT THEM TO STOP.
WE HAD ENOUGH SWEATERS.

https://m.xkcd.com/2324/

“Poorly documented legacy code leads to anger. Anger leads to hate. Hate leads to spaghetti code.”



https://scitools.com/legacy-code-your-journey-to-the-dark-side/

Disclaimer

This lecture contains materials from:

- Philip Koopman - CMU