"Good judgment comes from experience, and experience comes from bad judgment."

– Frederick P. Brooks
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
 Airbus A400M flight recorder data confirms "quality issue" in setup caused failure.

**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.
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?)
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?
Thank You
For Shopping At
Wilkins
SHOP 'n SAVE Express

----------------------------------

*** LOADED ***

SUCCESSFULLY

** ** 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

!"#$%^&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
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

Challenges
- Disaster recovery (includes vendor bankruptcy)
- End-of-life for hardware & software
- Adding new services to existing platform
- Security problems if not updated
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-doomsdaysubs-run-windows-xp/
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, ...
Writing

Useless Git Commit Messages

© 2020 Philip Koopman
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.

"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.