RE@21
Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems

Robyn Lutz
Dept. of Computer Science
Iowa State University, Ames IA
& formerly Jet Propulsion Lab, Pasadena, CA

RE'13, PUC-Rio de Janeiro, Brazil
July 17, 2013

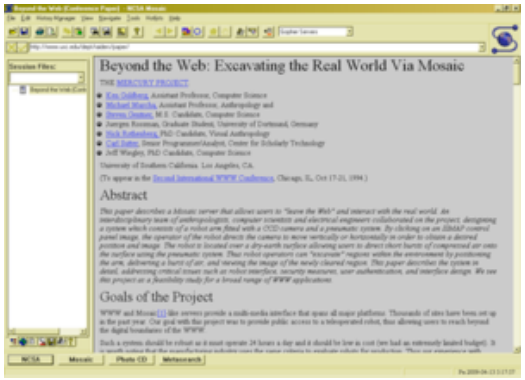
Some of this research was performed at ISU and supported by NSF 0541163, 0916275, & 1247051, and some was performed at JPL/Caltech and supported by NASA OSMA SARP.

Way back in 1993

sandiego.org

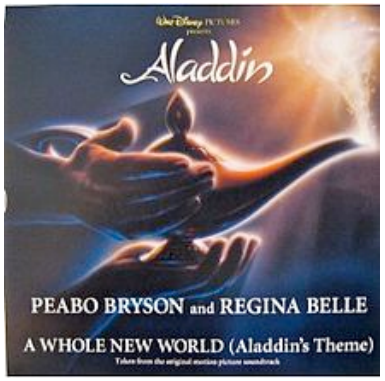
RE'93 is held in San Diego, CA

wikipedia



The Mosaic browser is released

The first Intel Pentium chips are shipped



wikipedia

"A Whole New World"
wins the Oscar for Best Song

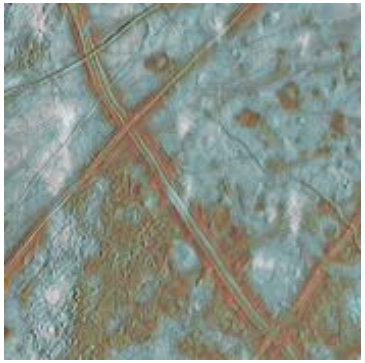
The first Beanie Babies appear

wikipedia

Old news now

Missing and misunderstood software requirements really do jeopardize safety-critical systems (spacecraft).

twitter.com



Approach & Findings in ‘93

- Examined 387 software errors discovered during integration & system testing on the Voyager and Galileo spacecraft, about half of which were safety-critical
- Used Nakajo & Kune [TSE’91] classification:
  software error ← human error ← process flaw

- Found prevalent error mechanisms producing safety-critical errors:
  - Missing requirements (e.g., for reasonableness checks on input data, due to incomplete documentation)
  - Misunderstood requirements (e.g., of hardware/software interfaces, due to lack of communication between teams)

Images courtesy of NASA/JPL-Caltech

More specifically

- "Difficulties with requirements is the key root cause of the safety-related software errors that have persisted until integration & system testing."
- Safety-related software errors have different causes from non-safety-related software errors
- We can make spacecraft safer by understanding & removing the prevalent causes of those critical errors

Example of a software requirements error (from Mars Reconnaissance Orbiter, launched 2005, still operational)

The transponder is the spacecraft receiver/transmitter used for telecommunications.

During system testing, a *false assumption* regarding the transponder was discovered, resulting in *new requirements knowledge*. It had been assumed that the transponder state always reflected the state of the carrier, i.e., locked or unlocked. However, it was found in system testing that these values could be temporarily out of synchronization when the carrier detection was transitioning between locked and unlocked.

The consequence was that the flight software requirement for fault-protection checking of the transponder telemetry had to be revised.

Example of why we care: Voyager

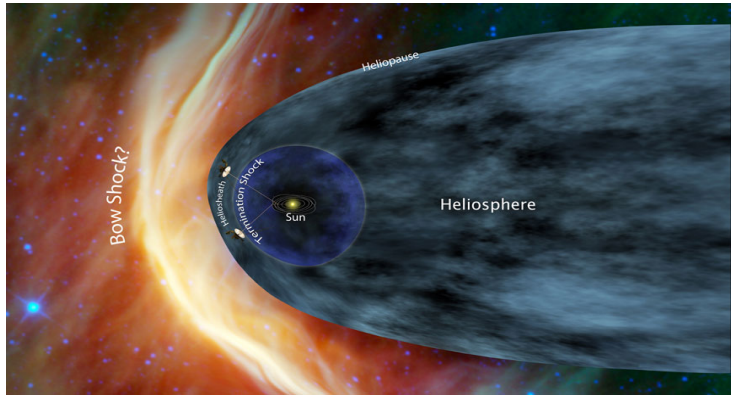
Long-lived: enduring, tenacious, robust

- Voyager 1 & 2 launched 1977: visited 4 planets
- Unprecedented discoveries (moons, rings, volcanoes, solar wind)
- Continue to be active science missions
- Farthest man-made objects (~11 billion miles)

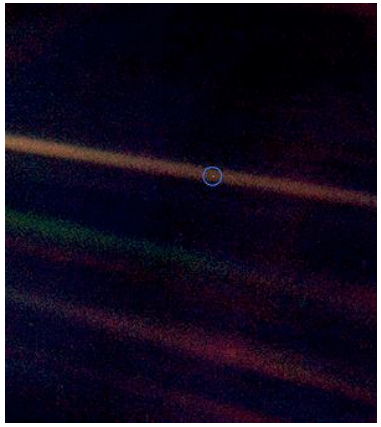
Nichelle Nichols, aka Lt. Uhura

Because *requirements change*:

- Original mission was extended to visit Uranus & Neptune, now planned to > 2020
- Failed receiver on Voy2 changed requirements
- Power depletion changes requirements



Images courtesy NASA/JPL-Caltech



Earth

A sturdy foundation

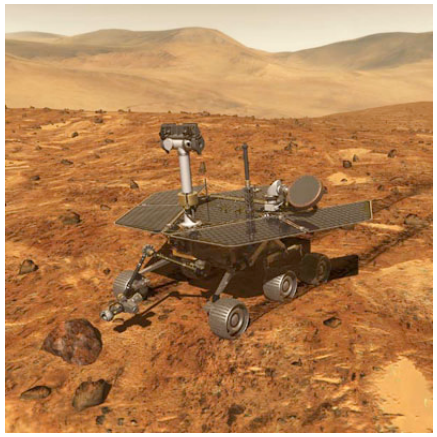
Some work that makes our systems safer:

cp-journal.com

- Letier and van Lamsweerde [TSE'00] obstacle analysis patterns
- Heimdahl [FOSE'07], called out difficulties with data-driven, configurable systems
- Habli and Kelly [SPLC'07] safety cases for product lines
- Jackson, Thomas, Millett, eds. ['07], report on software for dependable systems
- Strunk and Knight['08], assurance cases for dependable systems
- Hamill and Goseva-Popstojanova [TSE'09], requirements faults are common
- Miller, Whalen, Cofer [CACM'10] experience model checking safety-critical software
- Sabetzadeh et al. [HASE'10] focus on hardware/software interfaces
- Whittle, Sawyer, Bencomo, Cheng and Bruel , [REJ'10], requirements for self-adaptive systems
- Leveson, Engineering a Safer World ['12], broader organizational context
- Gay et al., ['12] optimization over requirements models
- Littlewood & Rushby [TSE'12], reliability claims

Our more recent work #1:

Analyzing software errors to understand ongoing *requirements discovery* in safety-critical systems



NASA/JPL-Caltech

Analyzed ~200 problem reports ranked critical on 7 launched spacecraft + ~450 on Mars Exploration Rover [Lutz & Mikulski, '04, '05].

Both sets showed 2 kinds of requirements discovery:

- Missing/new requirement, or unknown interaction
- Requirements confusion/misunderstanding by testers or operators

Both sets displayed 4 ways of handling requirements discovery:

- Software change
- Operational procedure change
- Document change
- No change (software behaved correctly)

Lessons learned:

- Plan for life-long RE
- Use reports of near-misses & false positives as a crystal ball
- Flag patterns of confusion
- Change software, not procedure
- Take a product-line perspective

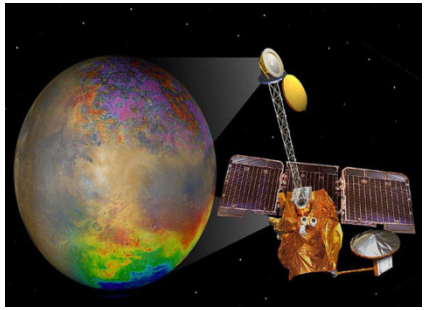
Our more recent work #2

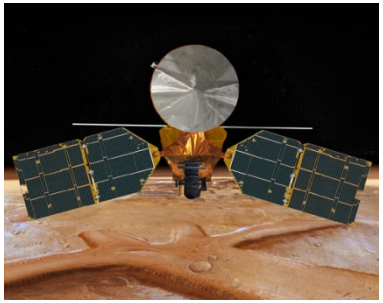
Software Requirements Errors in Safety-Critical Product Lines

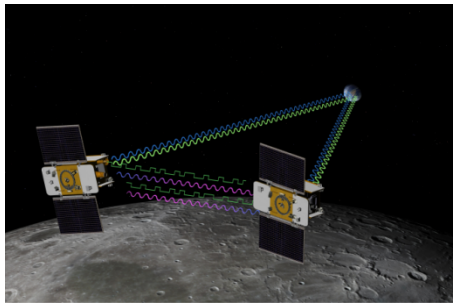
A flight-software product line has been used on 9 spacecraft managed by JPL, including GRAIL (mapped the moon in 2012) and Juno (launched 2011 for 5-year journey to Jupiter; Earth flyby for gravity assist on Oct. 9)

We used requirements information in problem reports from previous spacecraft to avoid recurrence of those problems on the next spacecraft. [Lutz, Lavin, Lux, Peters, Rouquette, 2013].

Lesson learned: Problem reports are a product line asset.







Images courtesy NASA/JPL-Caltech

Courtesy Classroom Clipart

Findings transfer to other high-integrity systems

- Rare events do occur & must be accounted for in the requirements
- Overly strict requirements unnecessarily add complexity & consequent risk
- Requirements always change after launch—and should
- Model-based engineering & good traceability reduce loss of knowledge over the lifetime of a long-lived system
- Using information in defect reports from previous product-line systems can reduce the next system's requirements-related defects
- Safety-focused checklists work & have the advantage that they tend to be updated
- There is no substitute for expertise & the power of a committed team

Siemens Medical

RE@21

- Missing and misunderstood software requirements still jeopardize safety-critical systems
- We still don't know much about the distinction in causes of safety-critical & non-safety-critical errors
- New kinds of safety-critical systems are bringing new RE challenges

  What does it mean to specify and validate requirements on programmable DNA self-assembled nanosystems?