SOFTWARE SAFETY

Gerard J. Holzmann

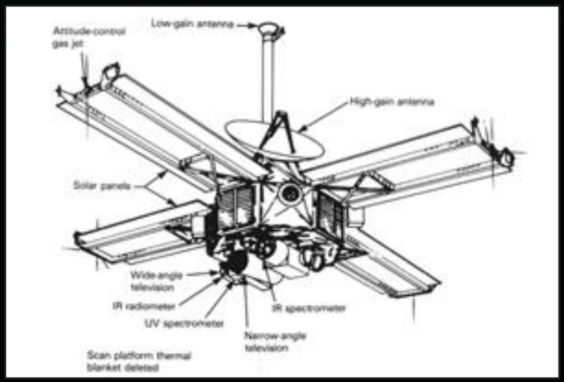
gh@jpl.nasa.gov

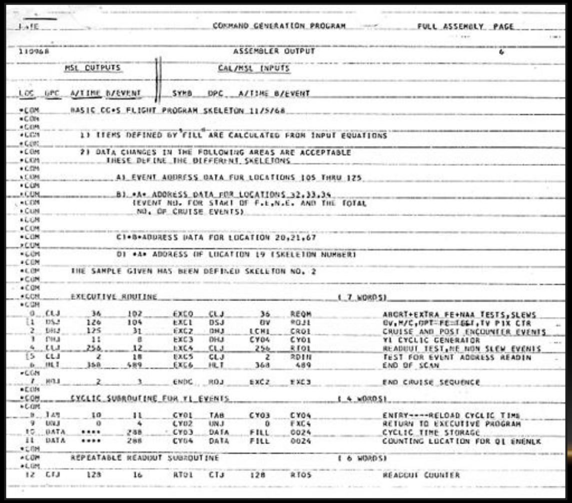
IS THERE A PROBLEM?

PROBLEM 1: SOFTWARE GROWS

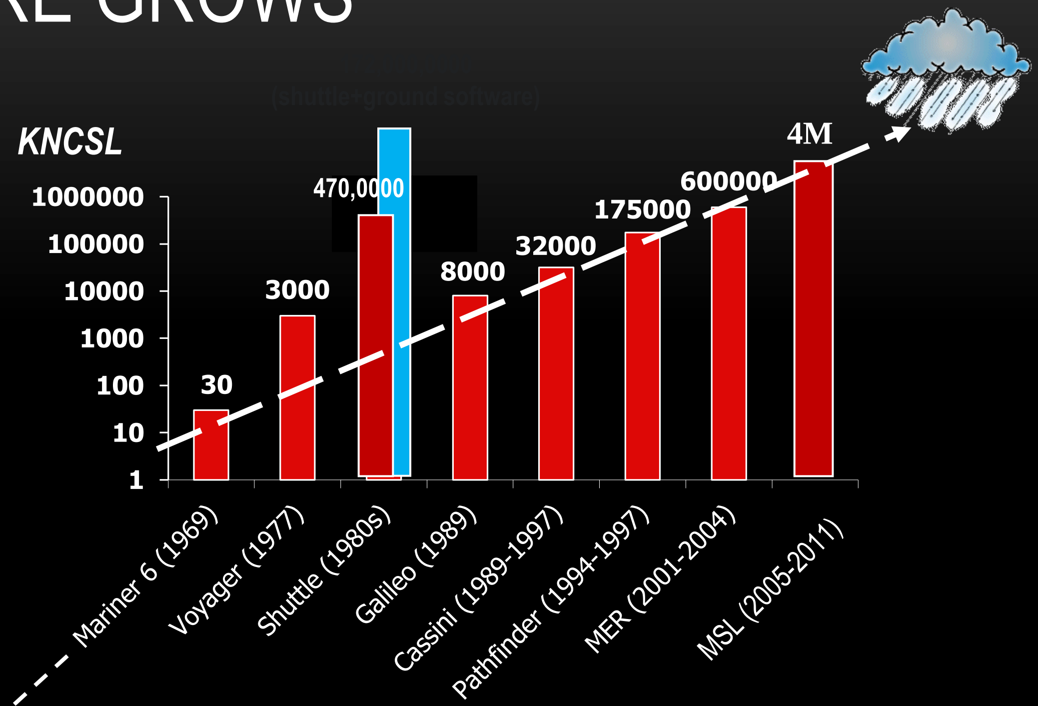
example: spacecraft control

Mariner 1969



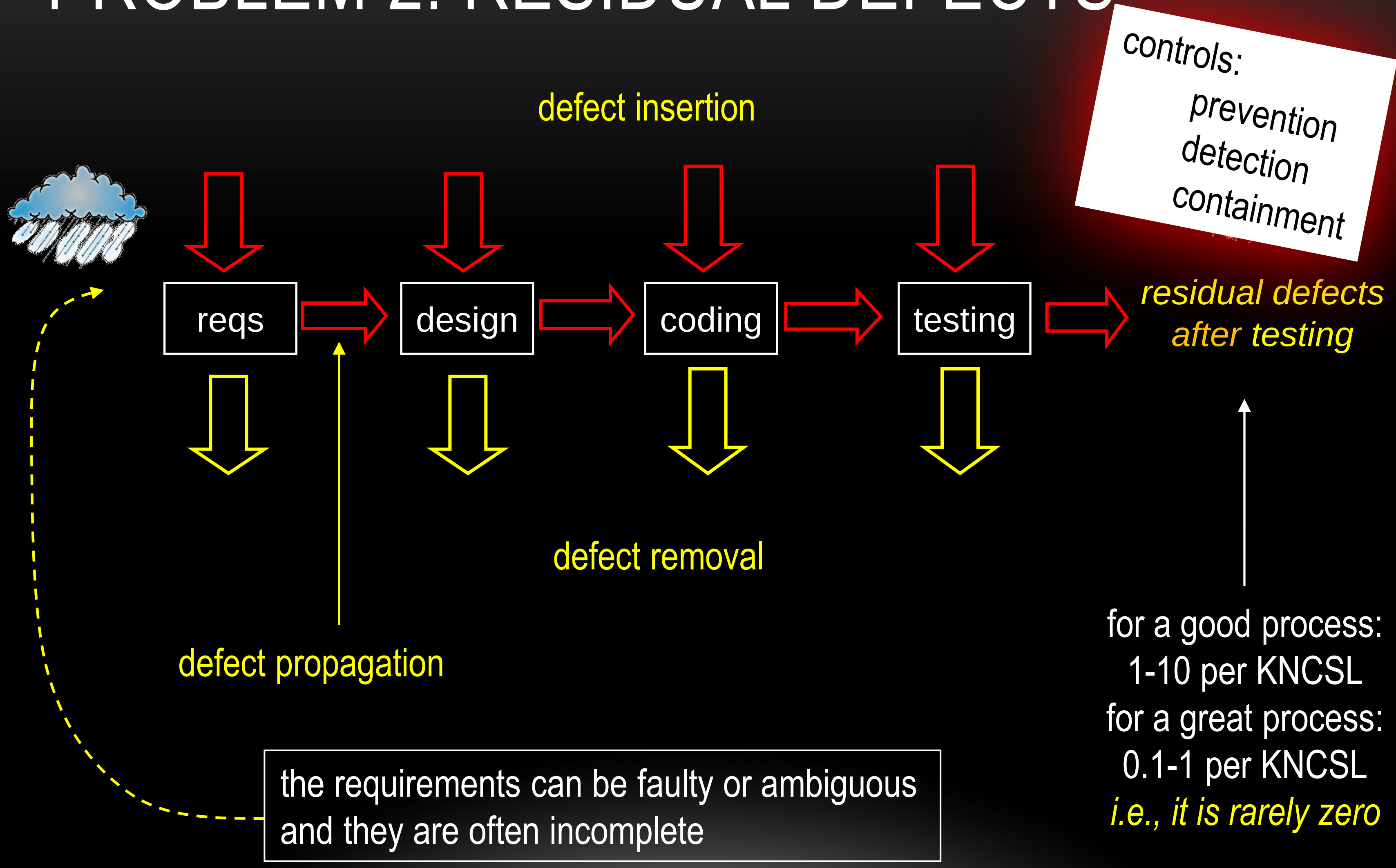


```
DATE                 COMMAND GENERATION PROGRAM          FULL ASSEMBLY  PAGE
110968                     ASSEMBLER OUTPUT                                6
        MSL CUTPUTS       ||        CAL/MSL INPUTS
LOC  OPC  A/TIME B/EVENT  ||   SYMB   OPC   A/TIME B/EVENT
*COM      BASIC CC*S FLIGHT PROGRAM SKELETON 11/5/68
*COM
*COM
*COM       1) ITEMS DEFINED BY 'FILL' ARE CALCULATED FROM INPUT EQUATIONS
*COM
*COM       2) DATA CHANGES IN THE FOLLOWING AREAS ARE ACCEPTABLE
*COM           THESE DEFINE THE DIFFERENT SKELETONS
*COM
*COM               A) EVENT ADDRESS DATA FOR LOCATIONS 105 THRU 125
*COM
*COM               B) *A* ADDRESS DATA FOR LOCATIONS 32,33,34
*COM                  (EVENT NO. FOR START OF F.E.N.E. AND THE TOTAL
*COM                   NO. OF CRUISE EVENTS)
*COM
*COM
*COM               C)*B*ADDRESS DATA FOR LOCATION 20,21,67
*COM
*COM               D) *A* ADDRESS OF LOCATION 19 (SKELETON NUMBER)
*COM
*COM      THE SAMPLE GIVEN HAS BEEN DEFINED SKELETON NO. 2
*COM
*COM
*COM      EXECUTIVE ROUTINE                                  ( 7 WORDS)
*COM
  0  CLJ    36   102   EXC0  CLJ     36  REQM        ABORT+EXTRA FE+NAA TESTS,SLEWS
  1  DSJ   126   104   EXC1  DSJ     OV  ROJ1        OV,M/C,OPT-FE-TEST,TV PIX CTR
  2  DHJ   125    31   EXC2  DHJ   LCH1  CRO1        CRUISE AND POST ENCOUNTER EVENTS
  3  DHJ    11     8   EXC3  DHJ   CY04  CY01        YI CYCLIC GENERATOR
  4  CLJ   256    12   EXC4  CLJ    256  RT01        READOUT TEST,NE NON SLEW EVENTS
  5  CLJ     2    18   EXC5  CLJ      2  RDIN        TEST FOR EVENT ADDRESS READIN
  6  HLT   368   489   EXC6  HLT    368   489        END OF SCAN
*COM
  7  ROJ     2     3   ENDC  ROJ   EXC2  EXC3        END CRUISE SEQUENCE
*COM
*COM      CYCLIC SUBROUTINE FOR YI EVENTS                    ( 4 WORDS)
*COM
  8  TAB    10    11   CY01  TAB   CY03  CY04        ENTRY----RELOAD CYCLIC TIME
  9  UNJ     0     4   CY02  UNJ      0  EXC4        RETURN TO EXECUTIVE PROGRAM
 10  DATA ****   288   CY03  DATA  FILL  0024        CYCLIC TIME STORAGE
 11  DATA ****   288   CY04  DATA  FILL  0024        COUNTING LOCATION FOR QI ENENLK
*COM
*COM      REPEATABLE READOUT SUBROUTINE                      ( 6 WORDS)
*COM
 12  CTJ   128    16   RT01  CTJ    128  RT05        READOUT COUNTER
```



the software for Mariner 6 (1969) was
128 words of assembler: equivalent to about 30 lines in C
(It had a backup control system in hardware.)

PROBLEM 2: RESIDUAL DEFECTS



PREVENTION, DETECTION, CONTAINMENT

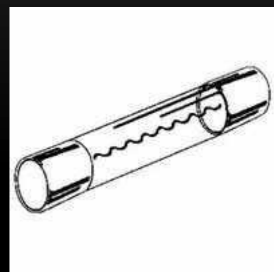
the process we introduced at JPL

- A lab-wide *coding standard* focused on *risk*-related rules
  - with *automated* compliance verification
- A software developer *certification* process
  - courses focused on SE principles and risk reduction
- A *senior managers* course, focused on software risk
  - many senior managers have limited exposure to software
- An emphasis on *tool-based analysis* (and not just people-based)
  - including tool-based *code review*
    - based on strong static source code analysis
    - and *daily* checks for coding-rule compliance
  - routine *logic model checking* for safety-critical parts of the design

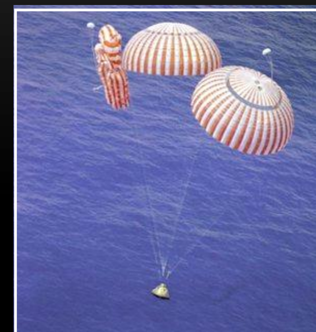

The Power of 10: Rules for Developing Safety-Critical Code

SYSTEM SAFETY

Design Principles: the first layer of defense



simplicity

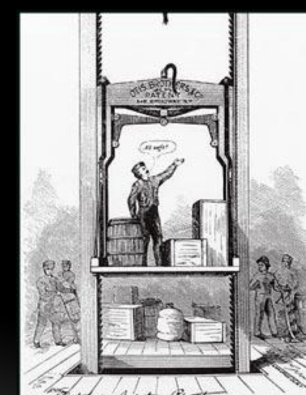


redundancy

diversity

Paranoia: the second layer of defense

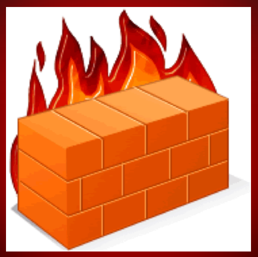
simplified backup



fault containment

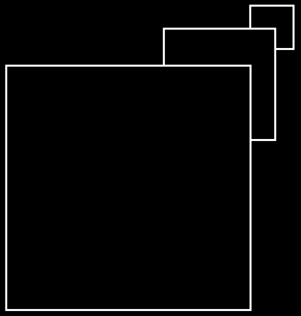
SOFTWARE SAFETY

- simplicity
  - software *modules* with well-defined rules for module composition; decoupling; fire-walls



- redundancy
  - emphasis on using *assertions*
  - *N*-version programming is of limited value

•J.C. Knight and N.G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-version Programming," *IEEE Trans. on SoftwareEngineering*,Vol. SE-12, No. 1 (Jan 1986), pp. 96-109.
•L. Sha. "Using Simplicity to Control Complexity," *IEEE Software*, July-August 2001, pp. 20-28.



- diversity
  - hierarchical *redundancy: hierarchies of increasingly simple and more strongly verifiable modules*

DO ASSERTIONS MAKE A DIFFERENCE?

Assessing the Relationship between Software Assertions and Code Quality: An Empirical Investigation

Gunnar Kudrjavets [1], Nachiappan Nagappan [2], Thomas Ball [2]
[1] *Microsoft Corporation, Redmond, WA 98052*
[2] *Microsoft Research, Redmond, WA 98052*
{gunnarku, nachin, tball} @microsoft.com

MSR-TR-2006-54

Abstract

The use of assertions in software development is thought to help produce quality software. Unfortunately, there is scant empirical evidence in commercial software systems for this argument to date. This paper presents an empirical case study of two commercial software components at Microsoft Corporation. The developers of these components systematically employed assertions, which allowed us to investigate the relationship between software assertions and code quality. We also compare the efficacy of assertions against that of popular bug finding techniques like source code static analysis tools. We observe from our case study that with an increase in the assertion density in a file there is a statistically significant decrease in fault density. Further, the usage of software assertions in these components found a large percentage of the faults in the bug database.

Keywords: Assertions, Faults, Bug database, Source control systems, Correlations.

1. Introduction

There is much literature that makes a case for the use of assertions and discusses the potential benefits of using assertions in software development. But to date, there have been limited studies in academia or in industry that empirically address the utility of assertions. Even when we talk to developers within Microsoft there are no unified opinions about the usefulness of assertions.

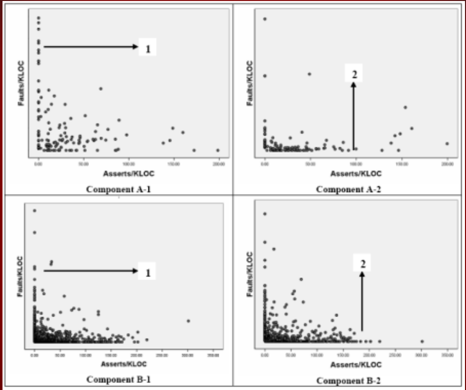
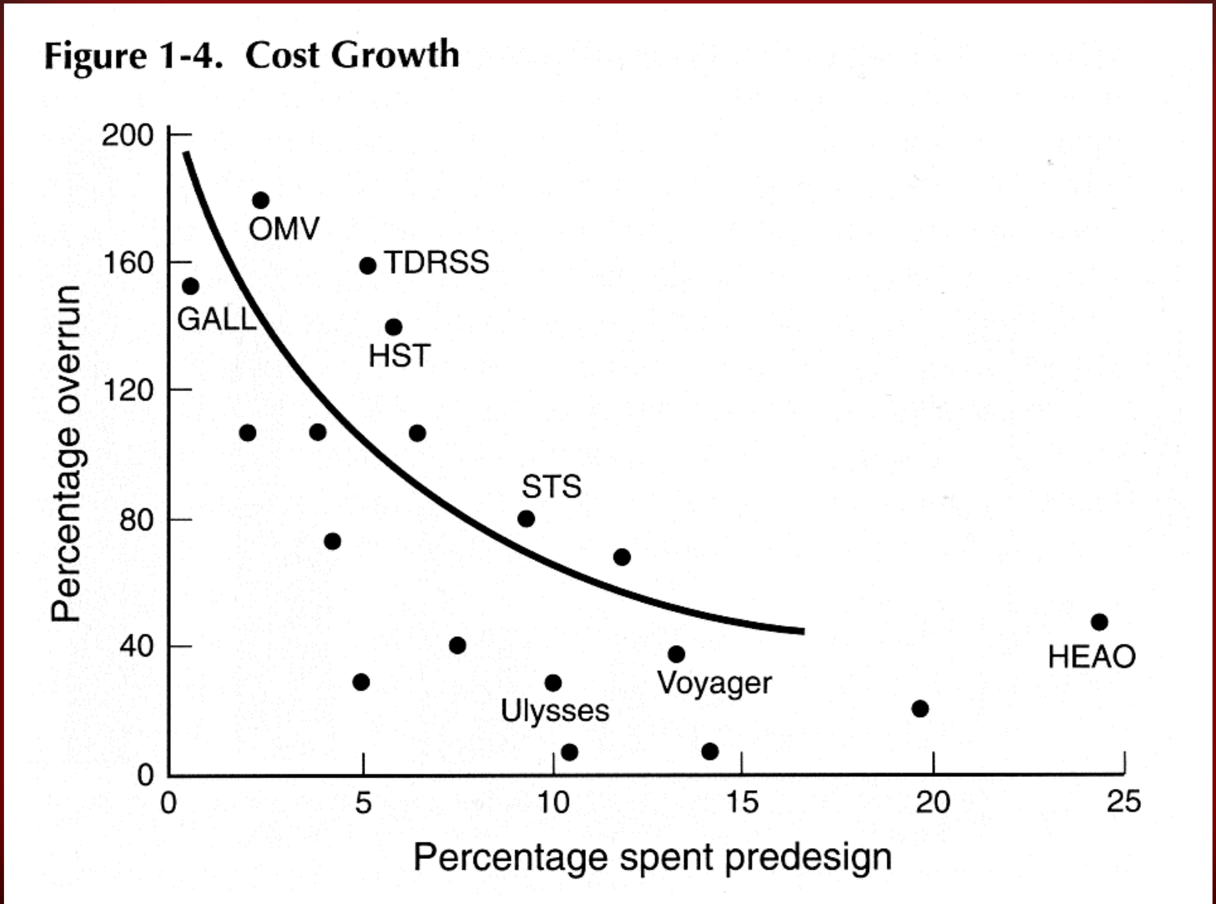


Figure 3: Scatter plots between assertion density and fault density for components A, B

"with an increase in assertion density there is a statistically significant decrease in fault density"

There are 250,000 assertions in the Microsoft Office source code (25M SLOC) i.e., 1% of the code [C.A.R. Hoare2003]

AN OUNCE OF PREVENTION



Source: "Customer-centered products – creating successful products through smart requirements management," Ivy F. Hooks & Kristin A. Farry, Amacom, NY, 2001, 272 pgs, ISBN 13-978-0-8144-0568-0

"The difference between a thing that can break and a thing that can't break is that when the thing that can't break breaks then it can't be fixed."

(Hitchhiker's Guide to the Galaxy, Book 5)

LARS

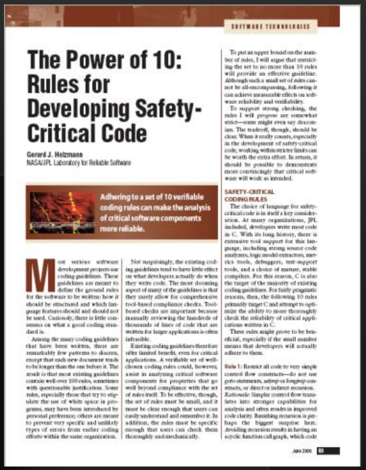
NASA/JPL Laboratory for Reliable Software

THE JPL CODING STANDARD FOR C

LEVELS OF COMPLIANCE

- LOC-1: language compliance
- LOC-2: predictable execution
- LOC-3: defensive coding
- LOC-4: code clarity
- LOC-5: MISRA *shall* compliance
- LOC-6: MISRA *should* compliance

THE POWER OF 10 RULES


SOFTWARE TECHNOLOGIES

The Power of 10: Rules for Developing Safety-Critical Code

Gerard J. Holzmann
NASA/JPL Laboratory for Reliable Software

Adhering to a set of 10 verifiable coding rules can make the analysis of critical software components more reliable.

SAFETY-CRITICAL CODING RULES

1. Restrict to simple control flow constructs
2. Do not use recursion and give all loops a fixed upper-bound
3. Do not use dynamic memory allocation after initialization
4. Limit functions to no more than ~60 lines of text
5. Use minimally two assertions per function on average
6. Declare data objects at the smallest possible level of scope
7. Check the return value of non-void functions; check the validity of parameters
8. Limit the use of the preprocessor to file inclusion and simple macros
9. Limit the use of pointers. Use no more than *N* level of dereferencing
10. Compile with all warnings enabled, and use source code analyzers

http://spinroot.com/p10/