Software Faults, Failures, and Fixes: Lessons Learned from a Large NASA Mission

Katerina Goseva-Popstojanova &
Maggie Hamill
West Virginia University
Morgantown, WV

Funding sources

- The work was funded in part by NASA OSMA SARP (2008) and NASA IV&V Facility (2009)

- Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NASA civil servants and contractors

Outline

- Introductory remarks
  - Motivation & Our goals & Benefits
  - Terminology
- Approach
- Project & data description
- Lessons learned
- Conclusion

Motivation & Our goal

- NASA spends time and effort to track problem reports/change data for every project. These repositories are rich, underused sources of information about the way software systems fail and types of software faults that cause these failures

- Our goal: Based on systematic and thorough analysis of the available empirical data, build quantitative and qualitative knowledge that contributes towards improving IV&V efficiency and software quality

Benefits

- The results of the systematic analysis of faults, failures, and fixes can be used to
  - Improve more cost-efficiently software quality
  - Improve the effectiveness of IV&V
  - Define metrics that prove the value of IV&V

- Additional insights related to improvement of the change tracking systems & quality of the data
  - Were reported to NASA
  - Are not included in this presentation

Collaboration with the IV&V team

- We collaborated closely with the IV&V team which provided invaluable support for our work
  - Domain knowledge
  - Feedback on data quality
  - Guidance & verification of database queries
  - Insights into significance and impact of the research results

- We reported the results to the IV&V team and incorporated their feedback into our work on a regular basis

Terminology

- **Failure**

  A departure of a system or system component behavior from its required behavior.

- **Fault**

  An accidental condition, which if encountered, may cause the system or system component to fail to perform as required.

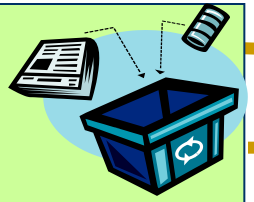
- **Fix**

  All changes made either to correct the fault(s) that caused an individual failure or to implement a workaround that prevents the failure from (re)occuring.

Approach

Explore change tracking system & Automate data extraction

Ensure data quality & Classify fault/failure/fix data



Analyze fault & failure attributes:

- Fault type
- Activity
- Severity

Analyze attributes of fixes

- Affected CSCIs
- Affected artifacts

Compile Lessons Learned & Recommendations for Improvements documents

Project: Basic facts

- A large, safety-critical NASA mission
  - Iterative development
  - Sustained engineering
- Implemented through Computer Software Configuration Items (CSCIs)
  - 21 CSCIs contain millions of lines of code in over 8,000 files
  - Developed at 2 to 4 locations
  - Span a wide range of applications

Data: Basic facts

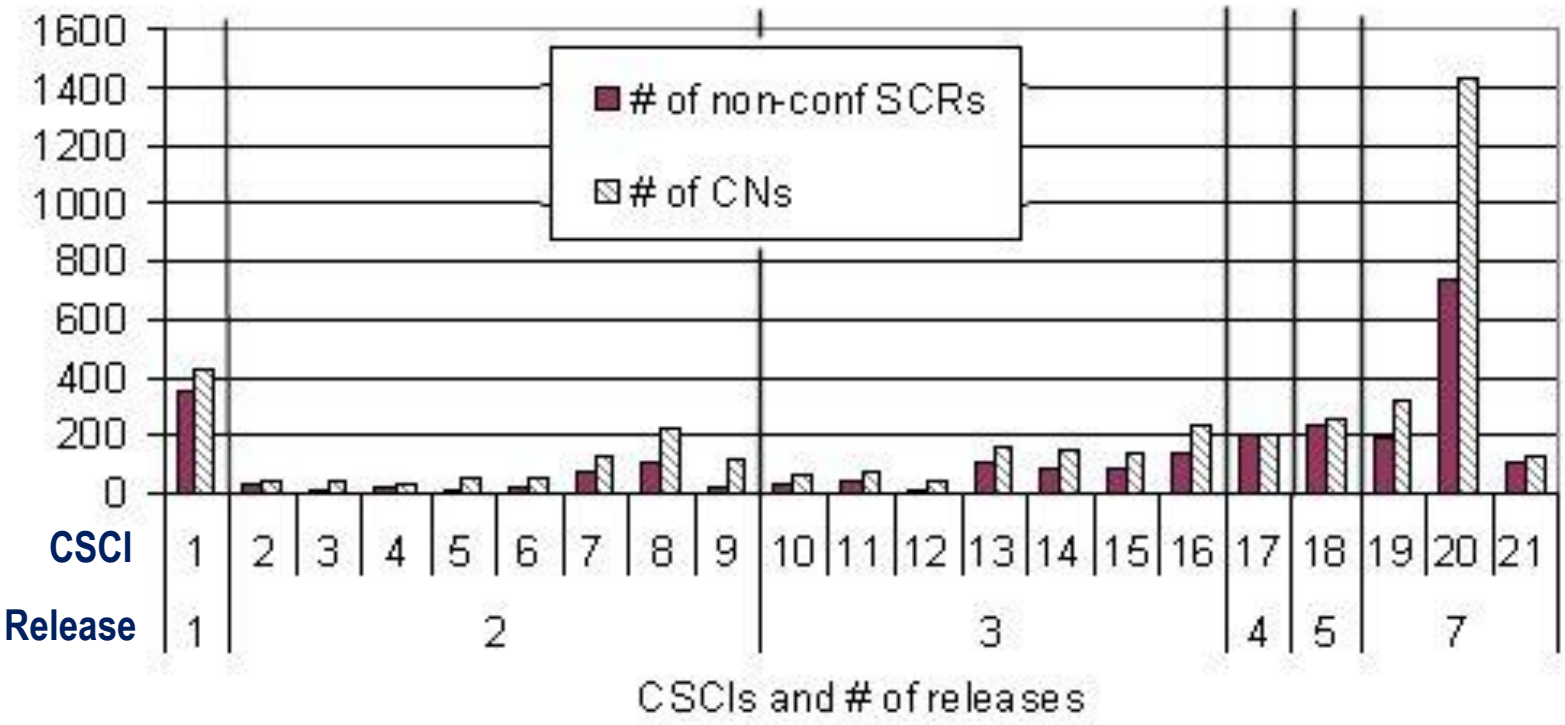
- We considered the Software Change Requests (SCR) which were created due to non-conformance to requirement(s)
  - An SCR represents either potential or observed failure reported throughout the life of each component
  - That is, while some of the failures were reported and addressed during development and testing, others occurred on-orbit

- Specifically, we considered
  - 2,500 non-conformance SCRs collected over 10 years
  - 4,700 Change Notices (CN) associated with these SCRs

LESSON 1:

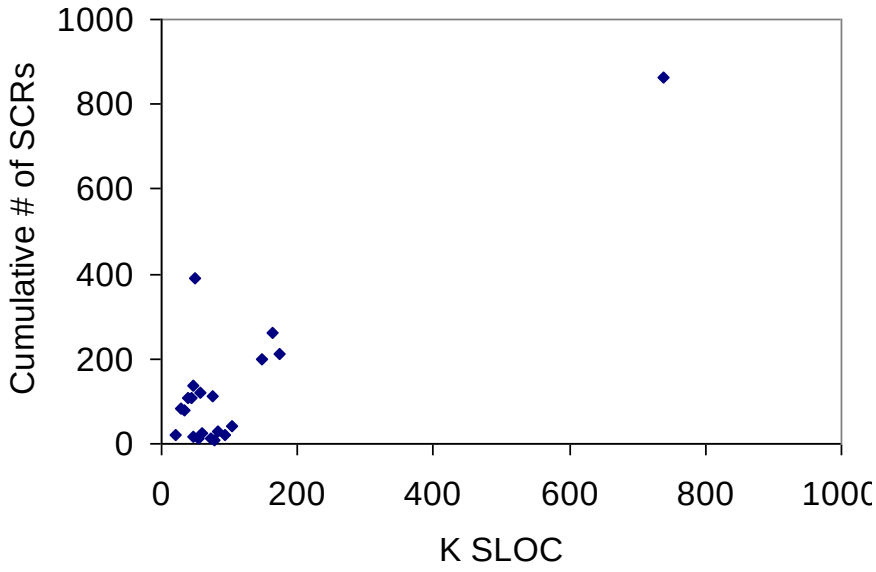
Larger CSCIs tend to have a larger number of non-conformance SCRs and larger number of CNs

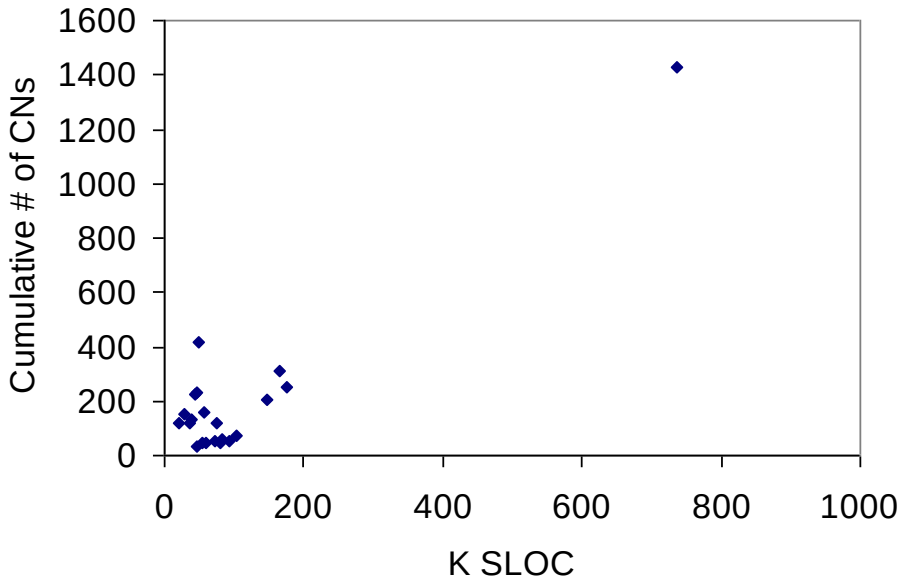
Lesson 1: SCRs and CNs per CSCI



	Min	Max	Median
# of SCRs per CSCI	8	735	75
# of CNs per CSCI	30	1430	121
# of CNs per SCR	0	41	1

Lesson 1: SCRs and CNs per CSCI



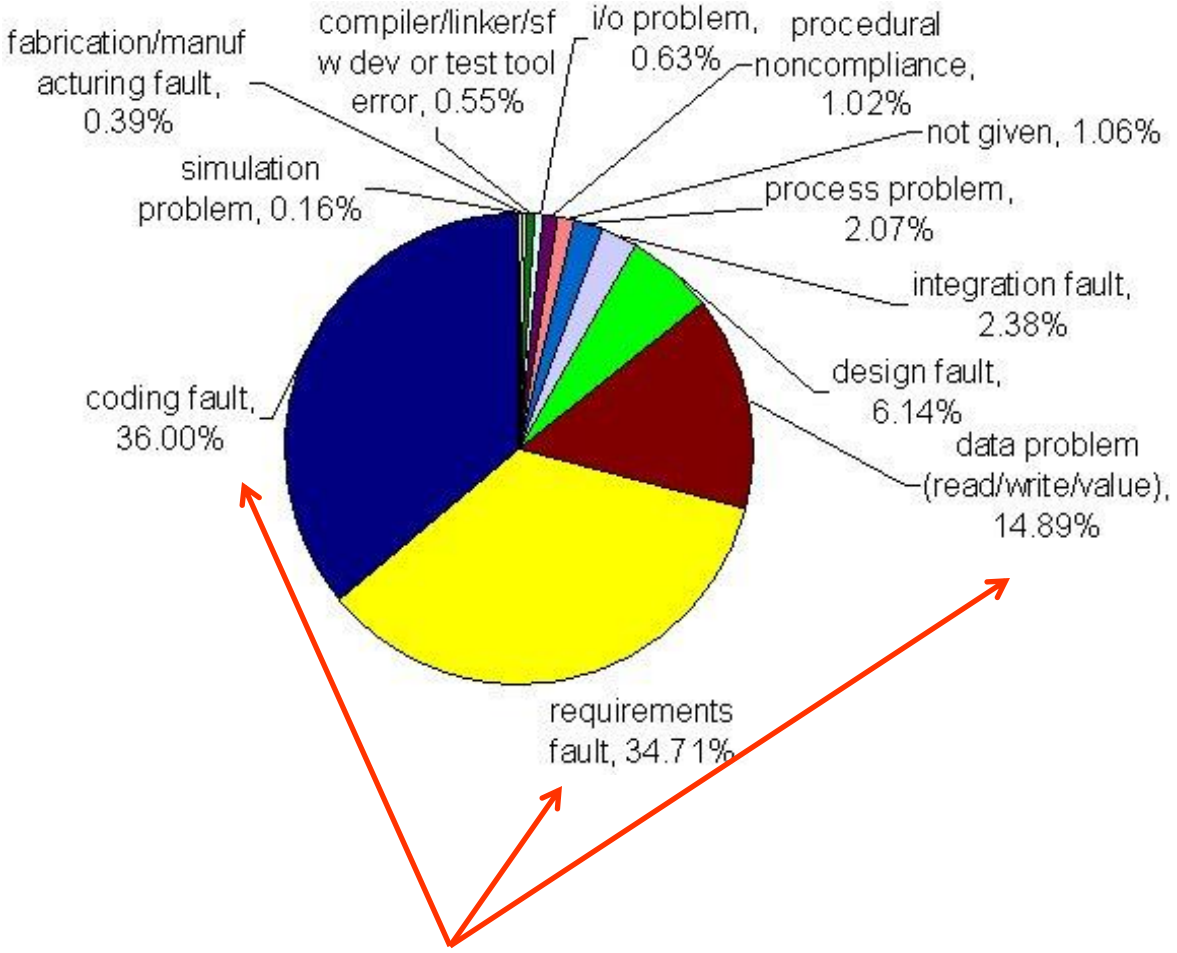


- Other factors, in addition to size, affect the non-conformance SCRs
- Outliers can be explained by the development history

LESSON 2:

Coding faults, requirement faults, and data problems are the dominating fault types

Lesson 2: Dominating fault types



Lesson 2: Dominating fault types

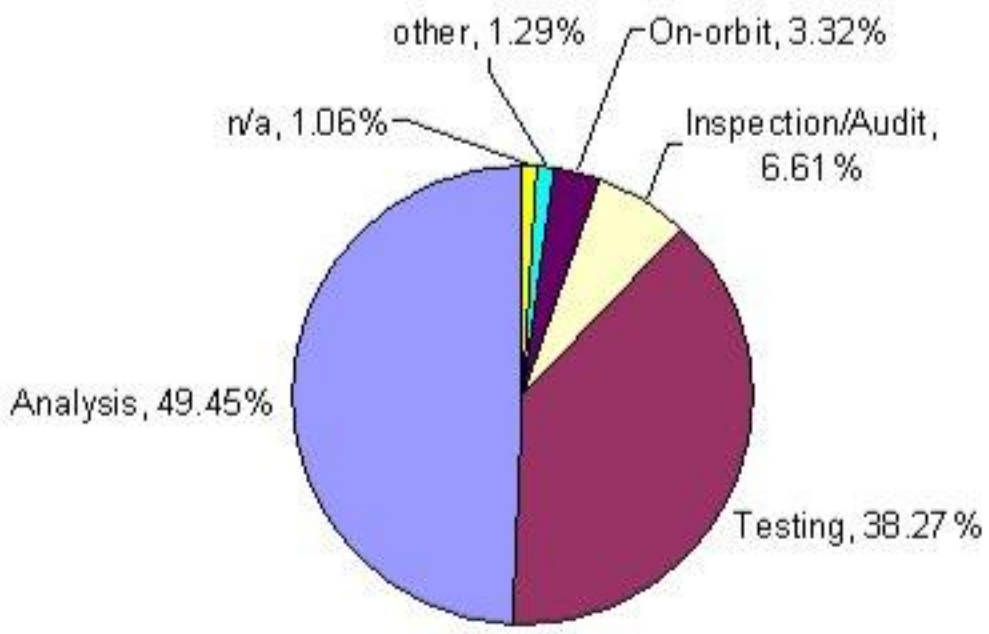
- Internal validity: Dominating fault types are consistent across SCRs grouped per release
  - 82-86% of SCRs in each release are due to the three dominating types of faults
- External validity: Results are consistent with several recent real world studies (Yu, 1998; Leszak et al., 2002; Lutz and Mikulski, 2004)
  - The percentage of the coding, interface, and integration faults is close to or even exceeds the percentage of requirements and design faults

LESSON 3:

Approximately half of all non-conformance SCRs were discovered by Analysis

Lesson 3: Detection activity

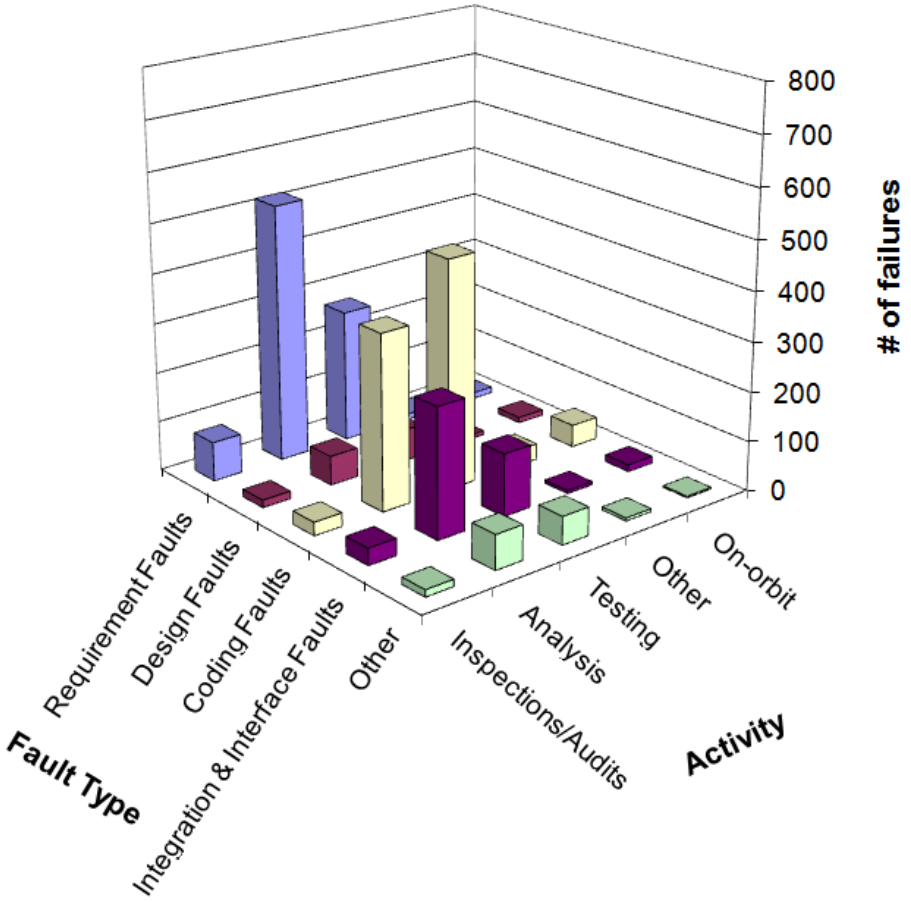
- Almost 50% of all non-conformance SCRs were discovered during Analysis, followed by 38% discovered by testing



LESSON 4:

Analysis was the most successful activity in detecting any type of faults. The only exception were coding faults for which testing did slightly better than analysis

Lesson 4: Activity & Fault type

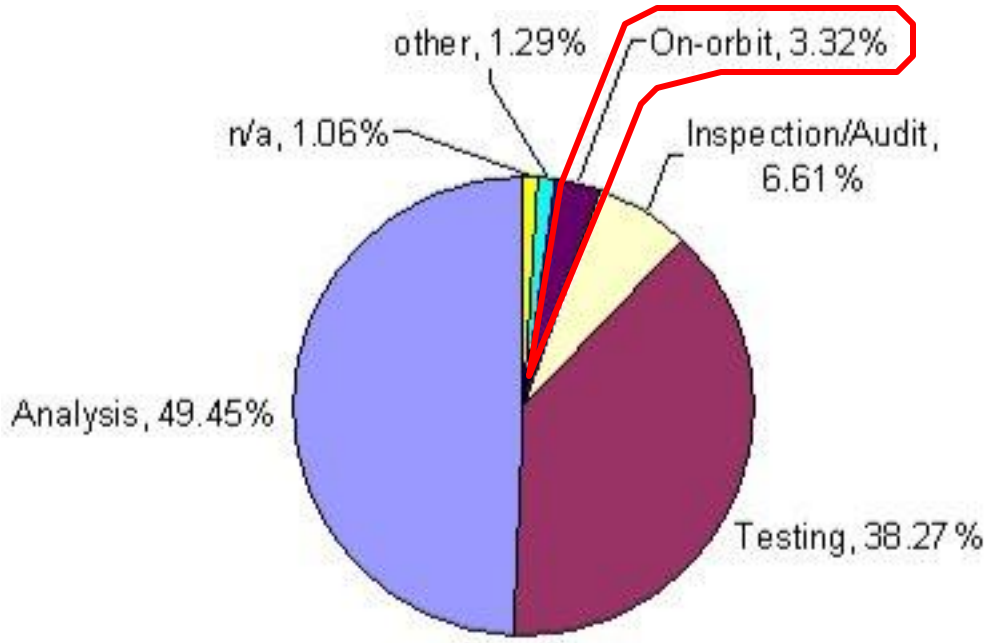


Analysis outperforms Testing for all fault types, except Coding faults (39% discovered by Analysis, 50% by testing)

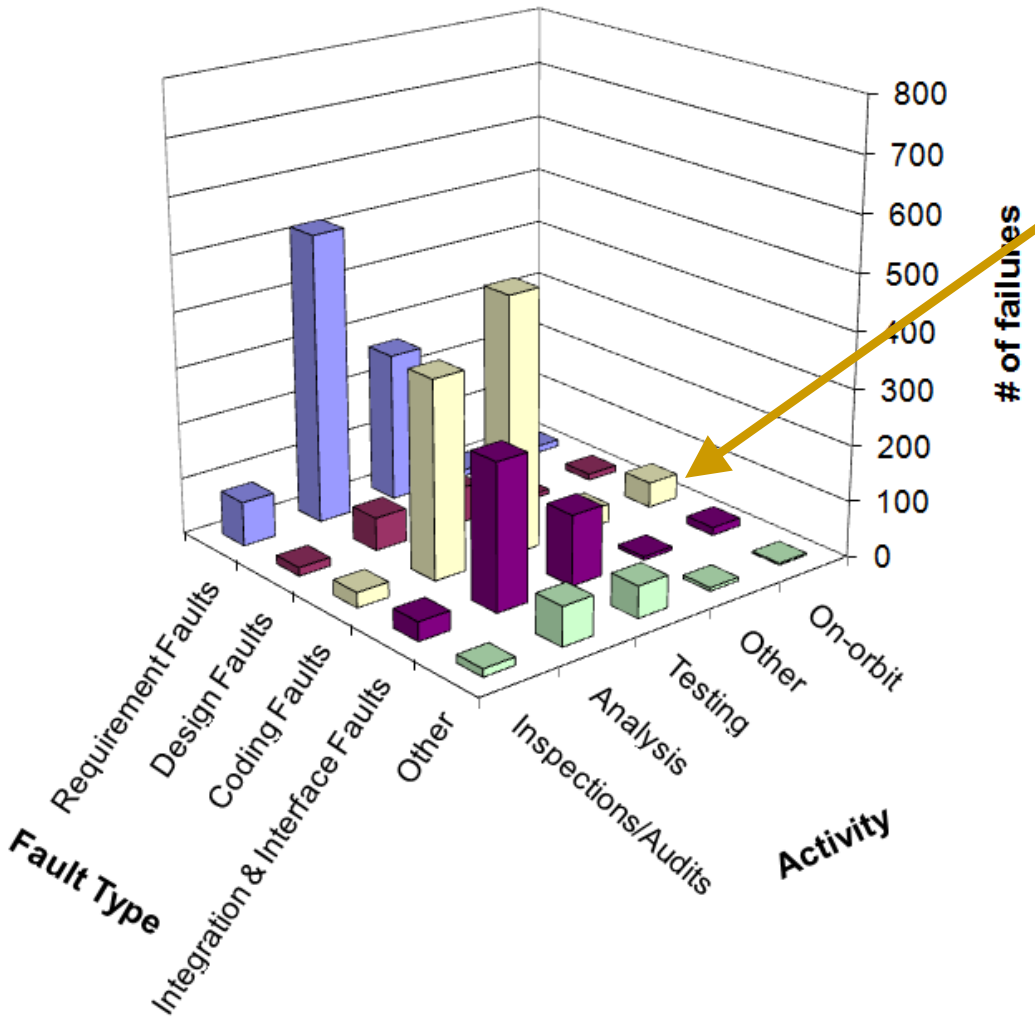
LESSON 5:

Only 3% of SCRs were detected on orbit. Over half of on-orbit SCRs were caused by coding faults

Lesson 5: On-orbit detection



Lesson 5: On-orbit detection



53% of SCRs discovered on-orbit were due to coding faults

LESSON 6:

Distribution of fault types differs for development & testing SCRs and on-orbit SCRs

Lesson 6: On-orbit detection & Fault type

Fault Type	% of Development & Testing SCRs	% of On-orbit SCRs
Requirements fault	35.42	14.12
Coding fault	35.42	52.94
Data problem	15.20	5.88
Design fault	5.90	12.94
Process and Procedure	3.19	0.00
Integration fault	2.10	10.59
Other	1.66	3.53
Not Identified	1.09	0.00

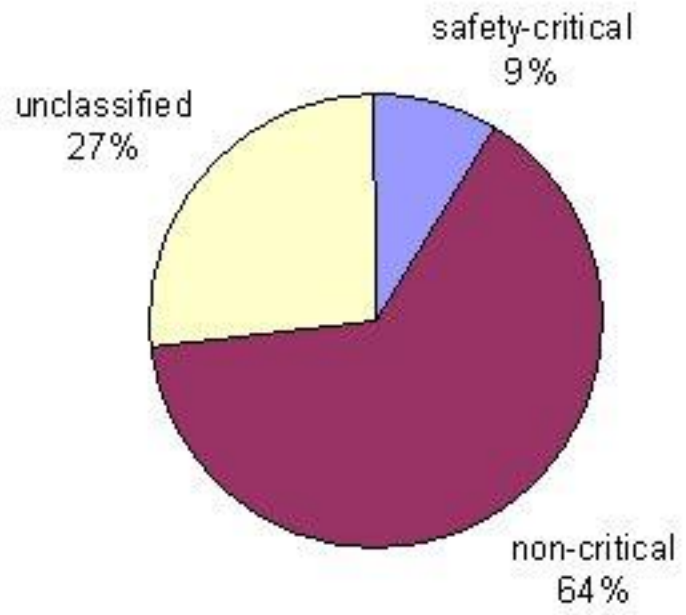
Relative % coding, design, and integration faults increase
Relative % of requirements faults and data problems decrease

LESSON 7:

Safety-critical failures are not frequent (less than 9%), most likely to be caused by coding faults, and almost equally likely to be detected by testing and analysis

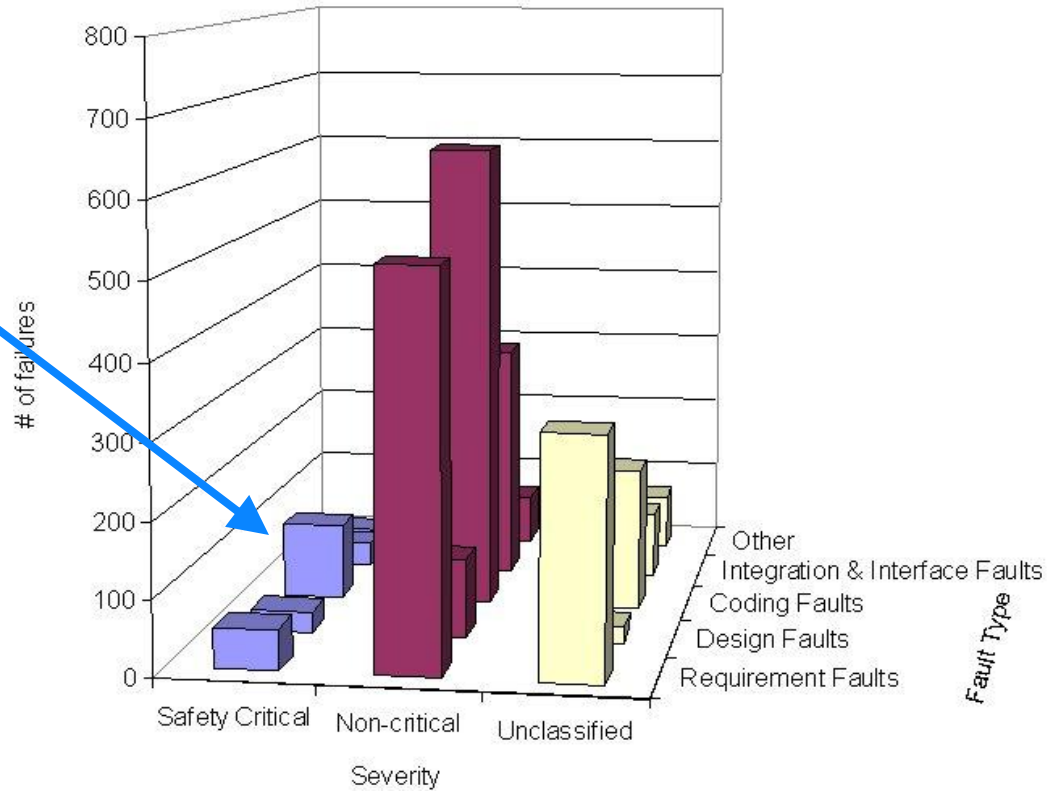
Lesson 7: Severity

- Safety-critical failures are less than 9%



Lesson 7: Severity & Fault types

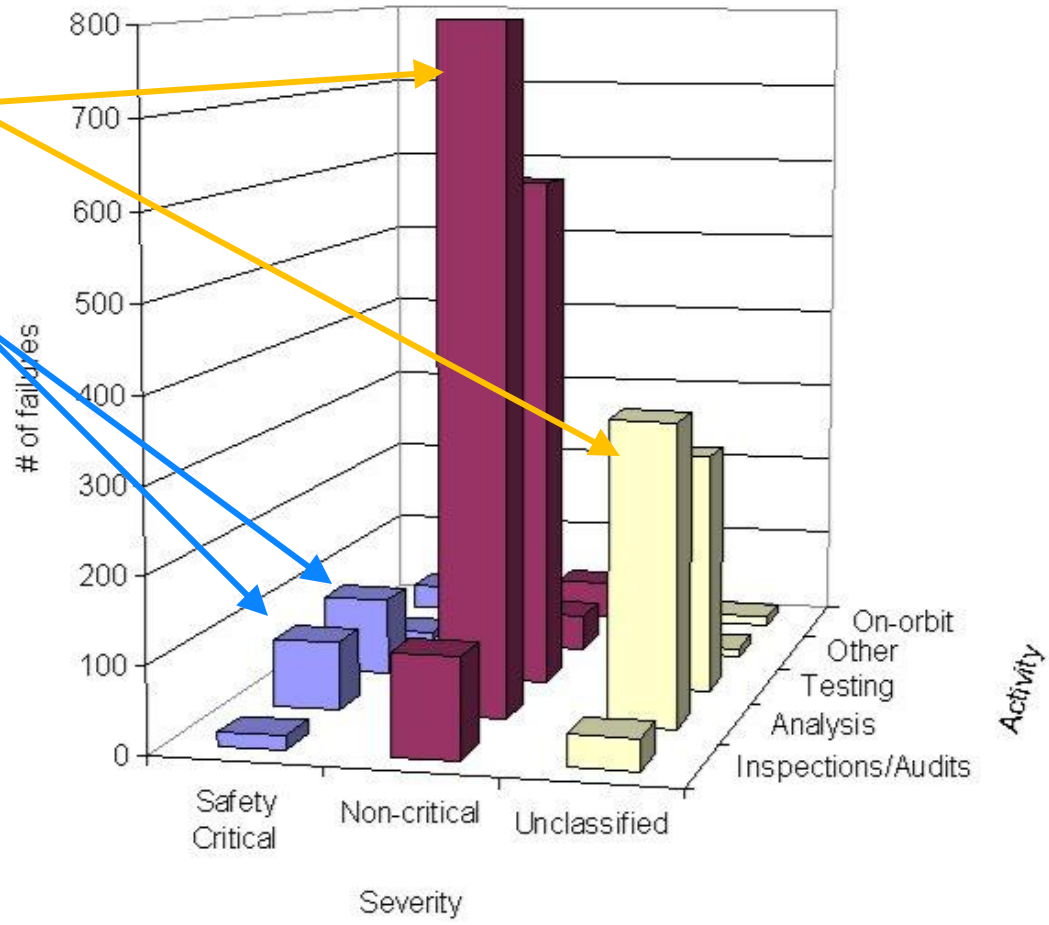
Coding faults are responsible for almost 50% of safety-critical failures!



Lesson 7: Severity & Activity

Analysis detected more failures than testing for non-critical and unclassified failures;

Testing performed slightly better for safety-critical failures! (40% vs. 35%)



LESSON 8:

The relative contribution of safety-critical failures on-orbit is higher than during development & testing

Lesson 8: Severity & On-orbit detection

Severity	% of Development & Testing SCRs	% of On-orbit SCRs
Safety–critical	7.93	34.12
Non–critical	64.50	52.94
Unclassified	27.58	12.94

Larger percentage of on-orbit failures are safety-critical

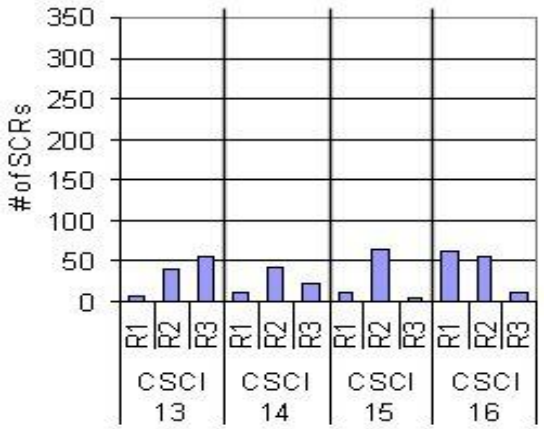
Greater emphasis is put on fixing and documenting on-orbit failures (only 13% of on-orbit failures remain unclassified)

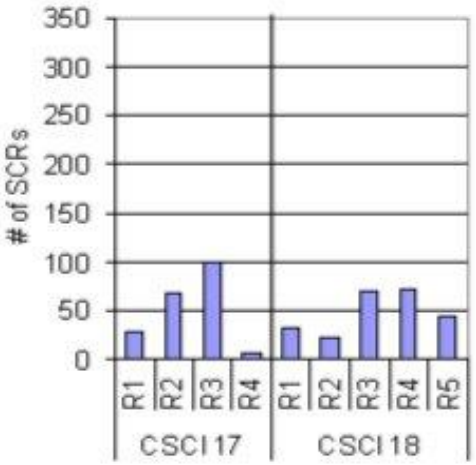
LESSON 9:

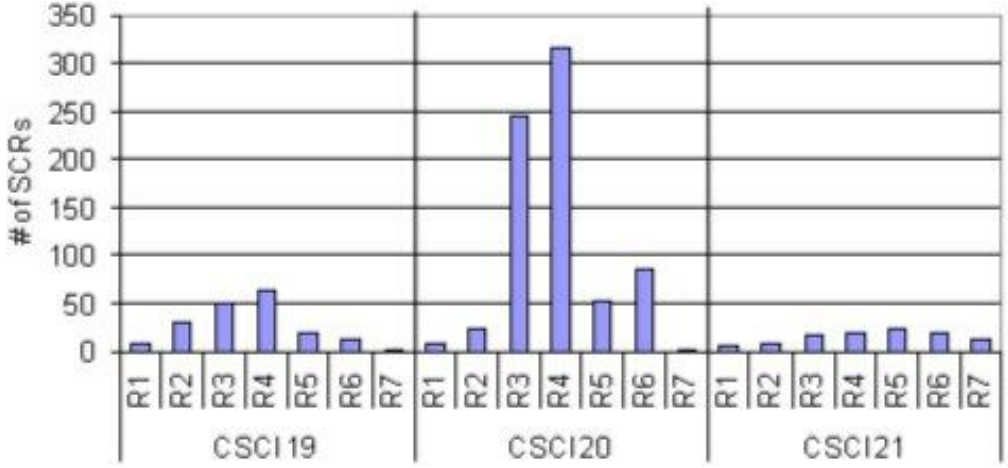
At the CSCI level, a larger number of non-conformance SCRs in one release does not necessarily imply a larger number in the subsequent release

Lesson 9: # of SCRs across releases

Subset of 11 CSCIs (7,8,13,14,15,16, 17,18,19,20,21) which have at least two releases and 70 non-conformance SCRs

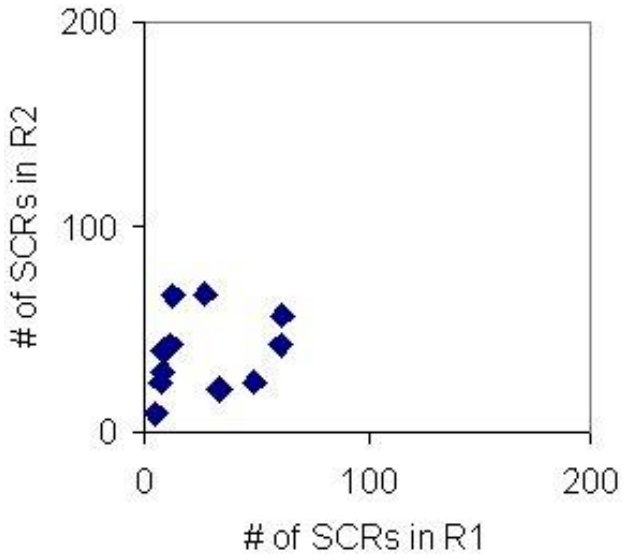


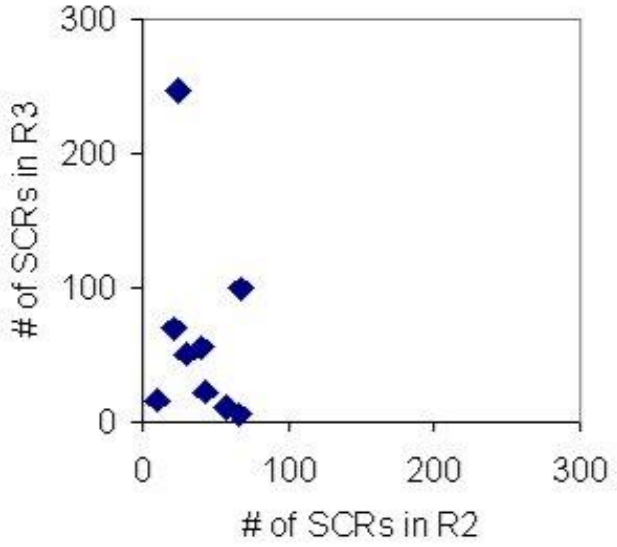




Bell-shaped curves show software reliability growth

Lesson 9: # of SCRs across releases





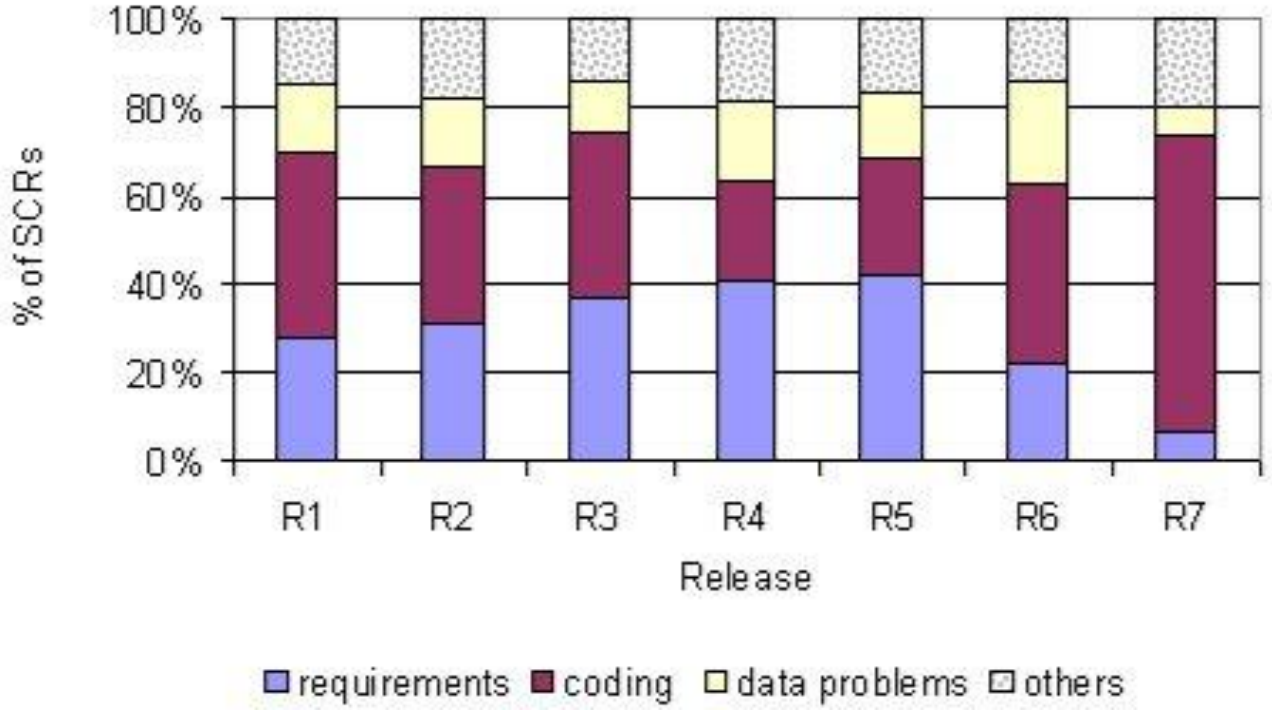
Number of SCRs in release n+1 is not correlated to the number of SCRs in release n

Results are not consistent with (Biyani and Santhanam, 1998; Ostrand and Weyuker, 2002; Pighin and Marzona, 2003)

LESSON 10:

At the CSCI level the dominant fault types (requirements faults, coding faults, and data problems) are consistent across releases

Lesson 10: Fault types across releases



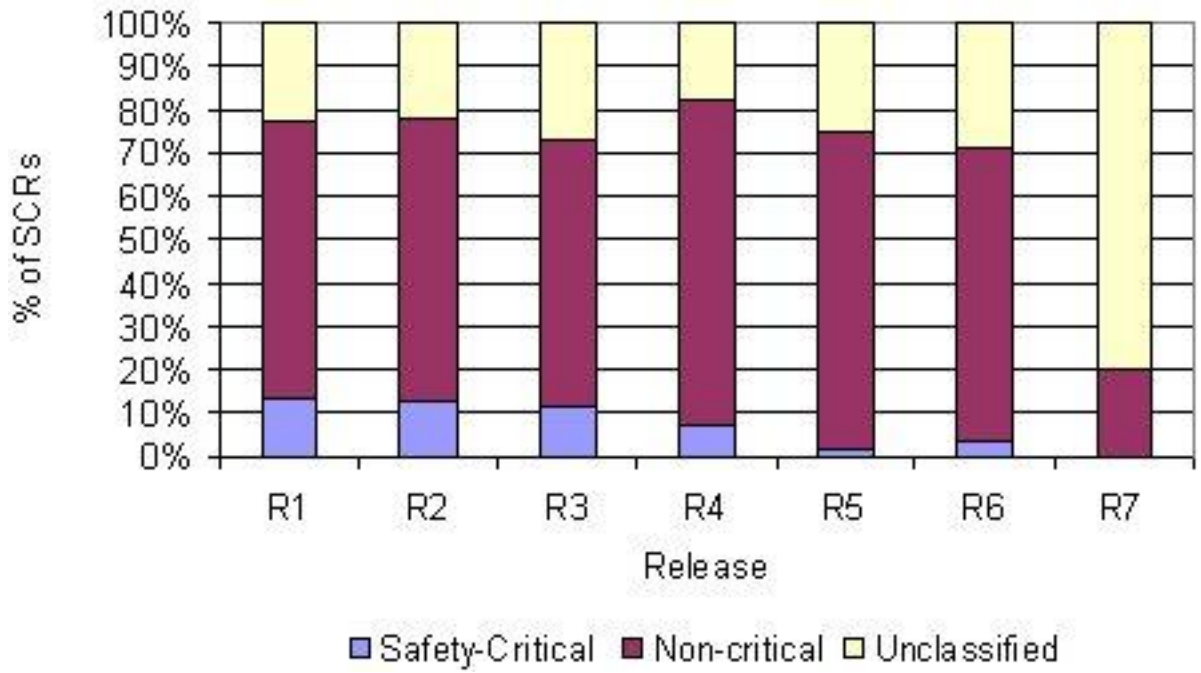
82-86% of SCRs in each release are due to the three dominating types of faults

For each CSCI, the vast majority of releases have at least 75% of failures caused by the common fault types: requirements faults, coding faults, and data problems

LESSON 11:

Safety-critical failures are not frequent in any release. Their occurrence tends to decrease as CSCIs mature

Lesson 11: Severity across releases



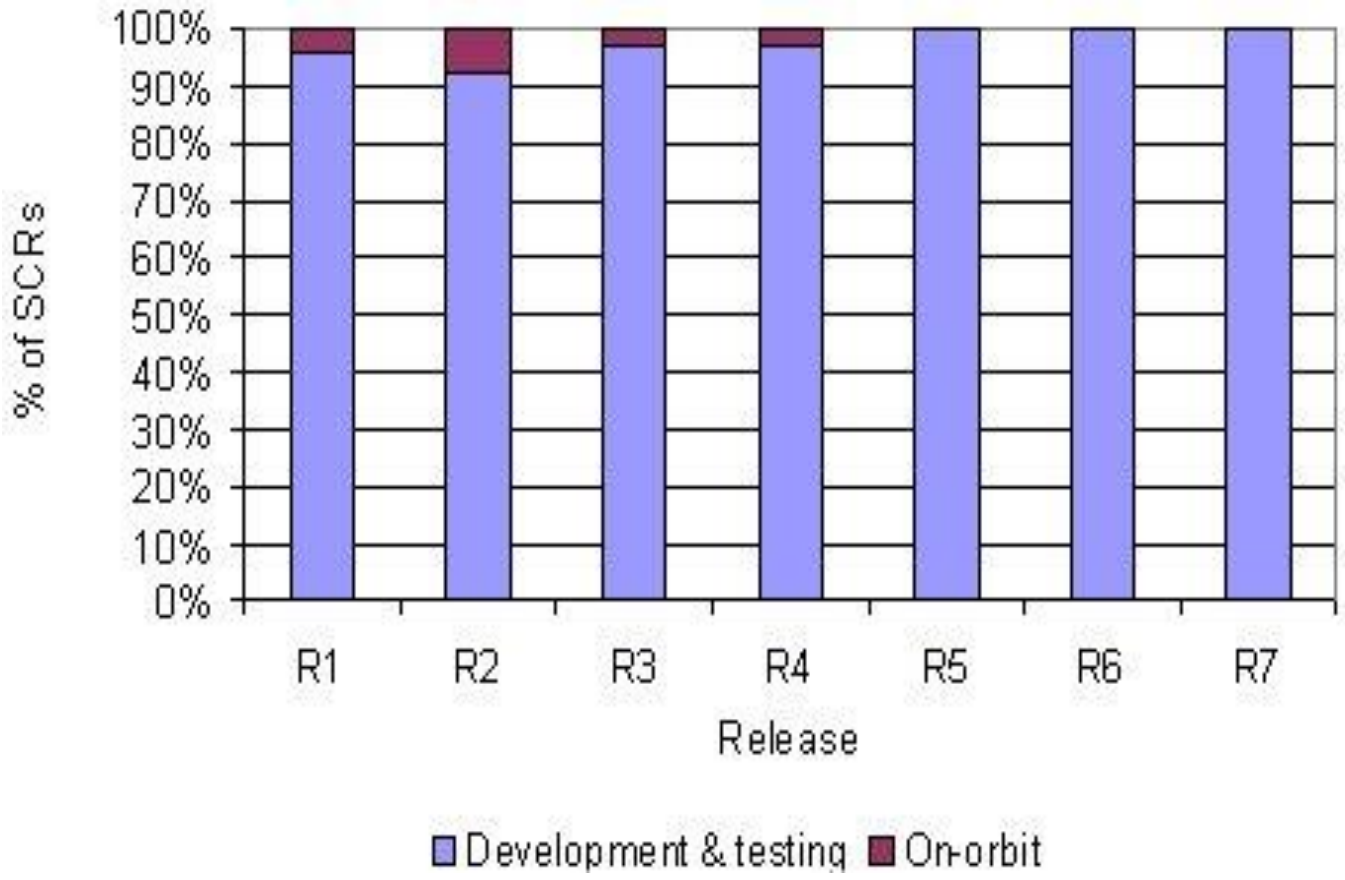
Cumulatively for all eleven CSCIs, 0% - 12% of non-conformance SCRs reported against each release were safety-critical

For each CSCI individually, on average less than 8% of SCRs reported against each release of that CSCI were safety-critical

LESSON 12:

On-orbit SCRs are rare in every release. Their occurrence tends to decrease as CSCIs mature

Lesson 12: On-orbit SCRs across releases

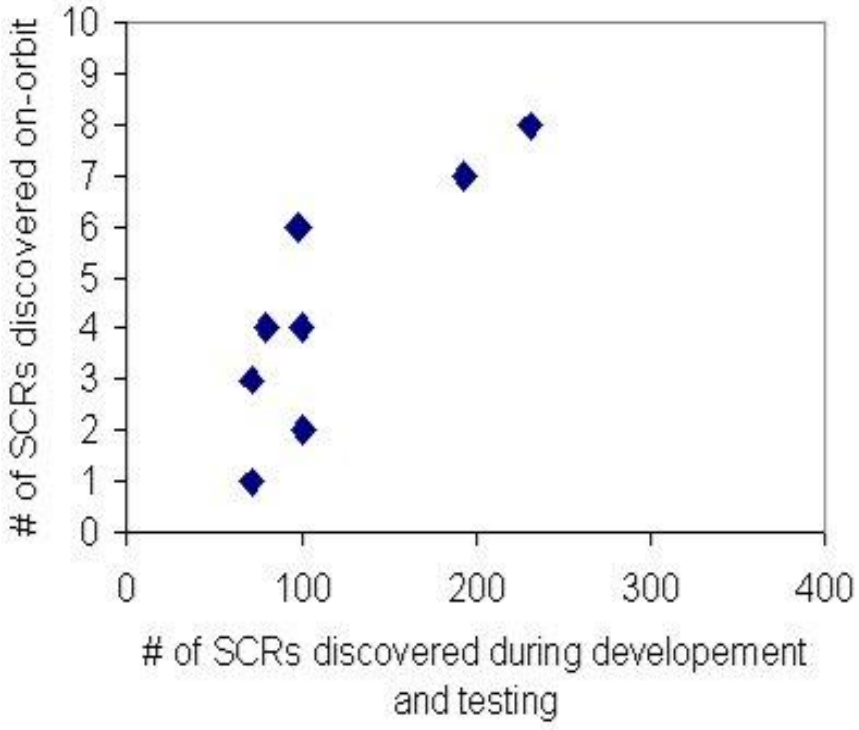


At the CSCI level, between 0% and 8% of failures per release occurred on-orbit

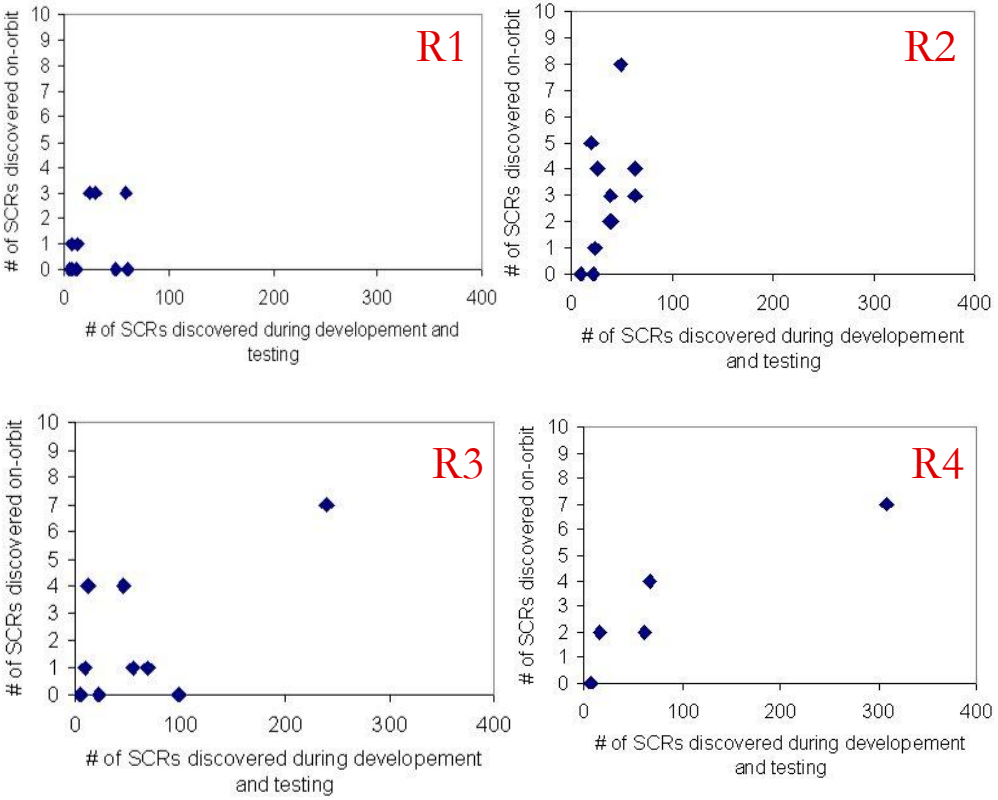
LESSON 13:

CSCIs with a larger number of SCRs discovered during development and testing tend to have larger number of SCRs on-orbit

Lesson 13: Persistence of failure proneness within release



Cumulative per CSCI



Per Release per CSCI

Results are consistent with (Andersson and Runeson, 2007), but not consistent with (Fenton and Ohlsson, 2000; Ostrand and Weyuker, 2002)

LESSON 14:

Individual SCRs are often associated with multiple fixes (i.e., changes) spread throughout the system

Lesson 14: Fixes are spread

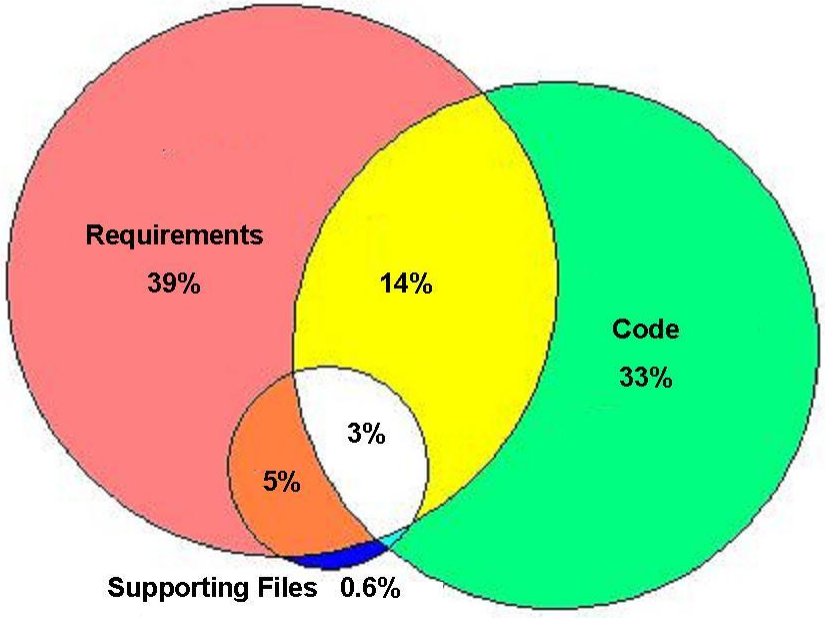
- For each SCR, we consider the failed CSCI (i.e., the CSCI against which the SCR was reported) and the fixed CSCIs (i.e., all CSCIs with changed artifacts as a result of this SCR)
  - 82% of SCRs led to fixes in only the failed CSCI
  - 15% of SCRs led to fixes in the failed CSCI and at least one other CSCI
  - 3% of SCRs lead to fixes only outside the failed CSCI
- Confirms our earlier work that single failures are often linked to multiple changes (Hamill and Goseva-Popstojanova, IEEE TSE, 2009)

LESSON 15:

The vast majority of SCRs resulted in changes to requirements, code, or the combination of both

Lesson 15: Affected artifacts

- Types of artifacts: requirements, design, code, supporting files, tools, notes/waivers
- 86% of SCRs caused changes in code (33%), requirements (39%), or the combination of both (14%)
- 2.5% linked to changes in design and at least one other type of artifact (not shown in the figure)



Conclusion

- Change Tracking Repositories provide a wealth of information on fault, failure, and fix data
  - It is important to enter high quality data
- Project Managers should be encouraged to explore both Developers' and IV&V Change Tracking Systems
  - Current practices (development and IV&V) can be improved based on the observed trends
- The internal and external validity indicate that several observed trends (e.g., dominating fault types) are not project specific. Rather, they seem to be intrinsic characteristics of software faults and failures. Other trends (e.g., failure persistence within and across releases) seem to be project specific.

Acknowledgements

We thank the following NASA civil servants and contractors. This work would not have been possible without their continuous help & support.

- Jill Broadwater
- Pete Cerna
- Susan Creasy
- Randolph Copeland
- James Dalton
- Bryan Fritch
- Nick Guerra
- John Hinkle
- Lynda Kelsoe
- Gary Lovstuen
- Tom Macaulay
- Debbie Miele
- Lisa Montgomery
- James Moon
- Don Ohi
- Chad Pokryzwa
- David Pruett
- Timothy Plew
- Scott Radabaugh
- David Soto
- Sarma Susarla