# FOUR THINGS THAT ARE ALMOST GUARANTEED TO REDUCE THE RELIABILITY OF A SOFTWARE INTENSIVE SYSTEM


Ann Marie Neufelder, Softrel, LLC

amneufelder@softrel.com

© Softrel, LLC 2020 This presentation may not be copied in part or in whole without written permission from AM Neufelder.

# Background

- Since 1987, these factors have been quantitatively associated with *more* reliable software
  - "Software Reliability, Measurement, and Testing Software Reliability and Test Integration" RL-TR-92-52, Rome Laboratory, Rome, NY, 1992
  - "The Cold Hard Truth About Reliable Software, Edition 6i", AM Neufelder, 2019.

| Factor associated with more reliable software | Examples |
|---|---|
| People | Domain experience, Team sizes and organization, geographical location, contract help versus employees, etc. |
| Processes | Degree to which software activities are defined and repeated |
| Techniques | Degree to which software engineers can develop software requirements, design, code, test plans that are most likely to meet requirements with fewest defects |
| Tools | Degree to which software organization can avoid tedious and repetitive tasks |

permission from AM Neufelder.

# Background

- Actual data from 140+ projects also shows that these factors are associated with *less* reliable software[1]

| Factor associated with less reliable software | Description |
|---|---|
| Size is grossly underestimated | Software size determines the schedule and the reliability prediction |
| Reliability growth is grossly overestimated | Reliability growth is how long the software version is tested in a real environment without added any new features |
| Defect Pileup | What happens when software releases are spaced too close together |
| Too many risky things happening in one software release | Risky things: New target hardware, version 1 software, brand new software staff, brand new software technology, brand new software processes or environments |

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.

# Size is grossly underestimated

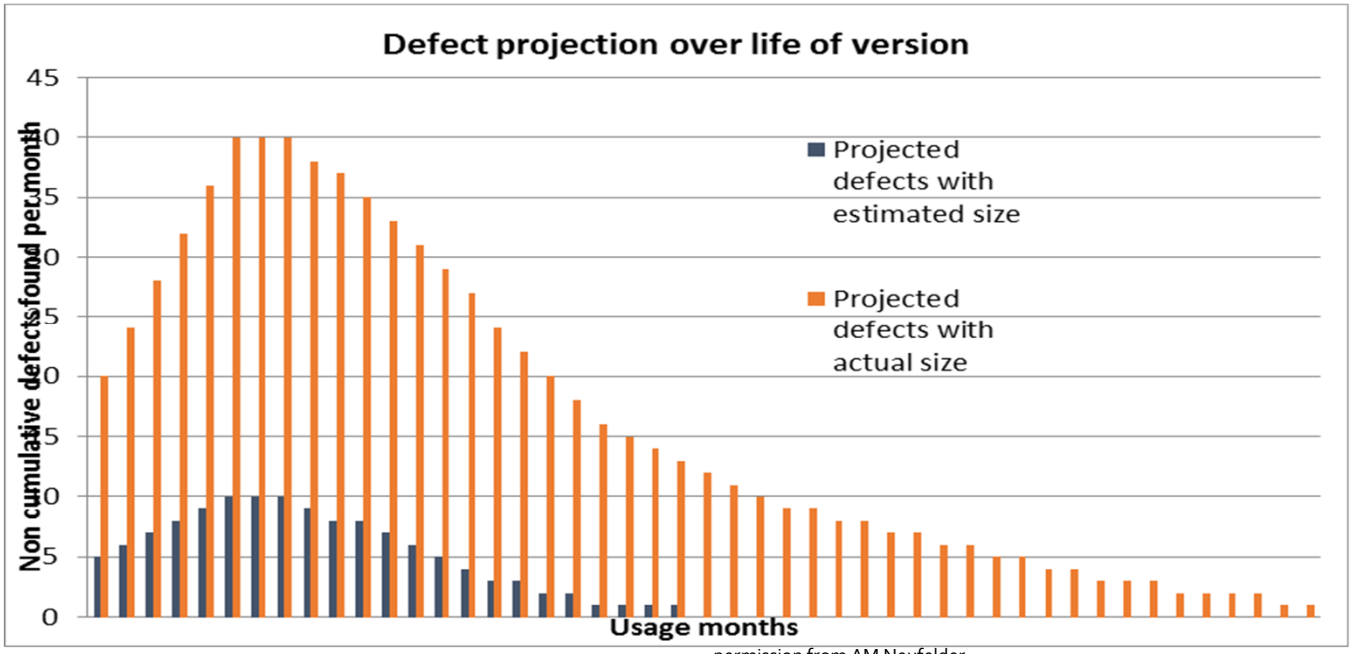
- *Theoretically* software size has linear relationship to reliability
  - Nearly all software reliability prediction models employ the below exponential formula to predict software failure rate [2]
  - Double the size -> Double the software faults -> Double the failure rate

| | |
|---|---|
| EKSLOC | Effective size of software in 1000 source lines of code |
| DD | Defect density – normalized operational defects per 1000 EKSLOC |
| Fo = EKSLOC * DD | Initial number of faults/defects in the code at delivery |
| K | Reliability growth constant related to number of deployed systems. |
| $N_o^{e-kti}$ | Number of faults/defects remaining in the code in the selected time period ti |
| $N_o^{e-kti-1}$ | Number of faults/defects remaining in the previous time period |
| $(N_{ti}- N_{ti-1})$ | Predicted software faults in between time i and i-1 |
| $(N_{ti}- N_{ti-1})/ t_i$ | Predicted software failure rate at month $t_i$ |

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.

# Size is grossly underestimated

- However, when size is *grossly* underestimated, effect on projected defects is *non-linearly* underestimated
  - Gross underestimates of size almost always result in serious schedule delay because there is more product to test and more defects to be found *than originally planned for*
  - Serious unexpected schedule delays usually mean less time for *software* reliability growth




permission from AM Neufelder.

# Software size can be *easily* overestimated when…

- Estimates of reuse are optimistic
  - If any of these things is true, the reused code estimate is probably optimistic
    - Reused code written in different language or development environment or Operating System
    - Reused code written for different target hardware
    - Reused code is more than a decade old
- Size estimates usually fail to consider that software has been growing in size at about 12% a year[2]
  - This is because software is replacing hardware functionality
  - Compare your kitchen appliances manufactured in 2014 to those from ten years ago!

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.

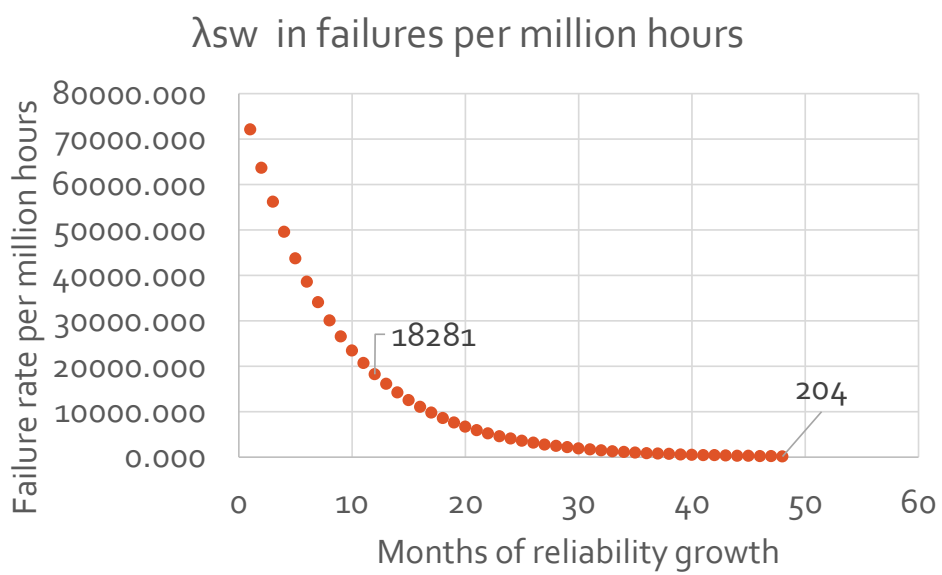
# Reliability growth is grossly overestimated

Reliability growth has been shown to have *non-linear* relationship to software reliability[3]

It can be grossly overestimated when…

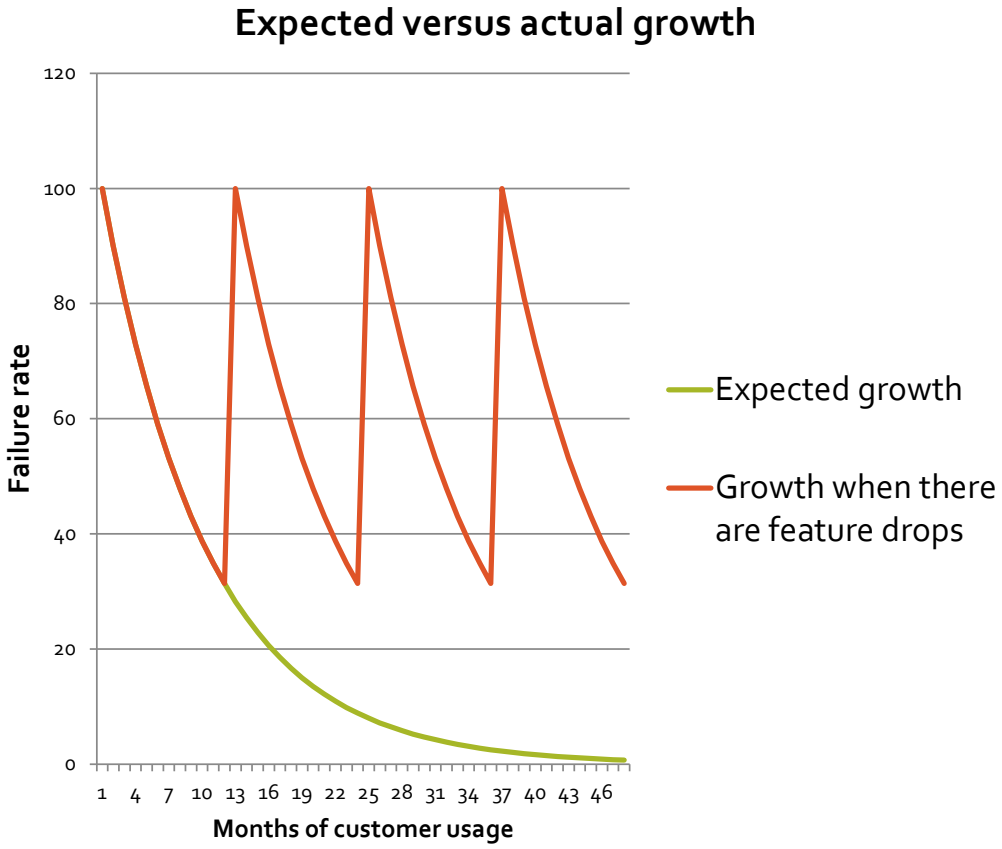
| | | |
|---|---|---|
| Size is grossly underestimated as shown previously | Unlimited reliability growth is assumed<br>Unless the software is at the end of its useful life it is virtually guaranteed that reliability growth is limited | If the overall schedule slips for software but not hardware, software will probably experience less reliability growth |

# How reliability growth has non-linear effect on failure rate




- Using industry accepted model shown previously…
- If software schedule slips such that ¼ of scheduled reliability growth can be experienced…
- The software failure rate can be about 90 times higher.

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.




# Reliability growth for software *is* bounded

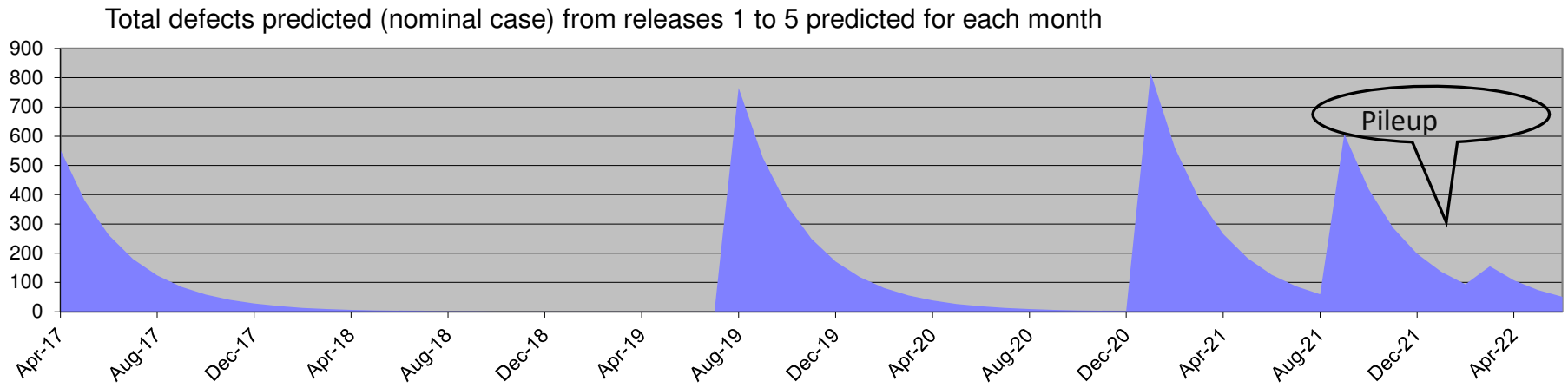
Common belief that software reliability grows indefinitely if software is used and defects are removed when discovered

This is only feasible if the software is at the end of it's useful life

Otherwise, it's almost certain to have future feature drops well before the failure rate bottoms out

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.

# Defects Can Pile Up




Defect pileup happens when new feature releases are too close together

Defect pileup is an extreme case of defect backlog – the backlog is increasing over time

This can easily happen if software defects are tracked or predicted in a vacuum or if they aren't predicted or tracked *at all*

Superimposing the cumulative predicted or actual defects as shown below is a simple but effective means to detect and avoid defect pileup

- These are serious risks that can impact the reliability of a software release
  - Substantially new target/system hardware
  - The software release is the very first version
  - The product or system is brand new
  - The software staff are new to the organization or industry
  - The software will employ a brand-new software technology
  - The organization will employ brand new or substantially modified software processes

Too many risky things in one release

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.

| | Successful release | Mediocre release | Distressed release |
|---|---|---|---|
| Fielded defect density (defects per normalized EKSLOC) | 0.04 | 0.31 | 1.63 |
| None of these risks existed for this field release | 78% | 27% | 0% |
| Exactly one of these risks existed for this field release | 11% | 64% | 50% |
| Exactly two of these risks existed for this field release | 11% | 6% | 30% |
| Exactly three of these risks existed for this field release | 0% | 0% | 10% |
| Four or more of these risks existed for this field release | 0% | 3% | 10% |

# Too many risky things in one software release

- Data from 140+ projects indicates that
  - Successful projects *never* had more than 2 of these risks
  - Distressed projects *always* had at least 1 risk

*The outcome of each project in the database was known to be either 1) successful 2) distressed or 3) neither. The third category is referred to as "mediocre".*

- *A successful project is defined as having a Defect Removal Efficiency (DRE) of at least 75% at deployment. None of these projects were recalled or cancelled.*
- *A distressed project is defined as having <= 40% defect removal at deployment. These projects were almost always recalled or cancelled.*

permission from AM Neufelder.

# Conclusions

- With regards to software reliability, most industry practices and standards focus on the processes, techniques, organization and tools *to achieve success*
- However, not as much is written about *how to avoid a distressed project*
- Distressed projects can be avoided if the causes for distress are spotted early
- Further reading
  - To read more about the practices that are related to successful, mediocre and distressed projects http://www.softrel.com/truth.htm
  - In 2016 a new book will be published with the entire set of factors that effects software reliability
  - The Software Reliability Toolkit training class and the Software Reliability toolkit also has the entire set of factors as well as a model to predict defects from those factors

© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.

# References


[1] A. Neufelder, "The Cold Hard Truth About Reliable Software, Edition 6i", Published by Softrel, LLC, 2019. http://www.softrel.com/truth.htm

[2] US General Accounting Office, "GAO report number GAO-10-706T entitled 'defense acquisitions: observations on weapon program performance and acquisition reforms' which was released on may 19, 2010. Http://www.Gao.Gov/products/GAO-10-706T

[3] Some references include

a) J. McCall, W. Randell, J. Dunham, L. Lauterbach, Software Reliability, Measurement, and Testing Software Reliability and Test Integration RL-TR-92-52, Rome Laboratory, Rome, NY, 1992

b) "System and Software Reliability Assurance Notebook", P. Lakey, Boeing Corp., A. Neufelder, produced for Rome Laboratory, 1997.

c) Keene, Dr. Samuel, Cole, G.F. "Gerry", "Reliability Growth of Fielded Software", Reliability Review, Vol 14, March 1994.


© Softrel, LLC 2020This presentation may not be copied in part or in whole without written permission from AM Neufelder.