Predict Software Reliability Before the Code is Written

Ann Marie Neufelder
Mission Ready Software
Sales@Missionreadysoftware.com

Missionreadysoftware.com

Software reliability timeline

1962 First recorded system failure

Many software reliability estimation models developed.
Main obstacle – can't be used until late in life cycle.

1968
The term "software reliability" is invented.

First publicly available model to predict software reliability early in lifecycle developed by USAF Rome Air Development Center with SAIC and Research Triangle Park –
Main obstacles – model only useful for aircraft and model never updated after 1992.

SoftRel, LLC develops models based on RL model but usable on all defense/space

A few proprietary models developed

1960's 1970's 1980's 1990's 2000's

Software reliability modeling

- Software reliability can be predicted before the code is written, estimated during testing, and calculated once the software is fielded
- This presentation will discuss the prediction/assessment models

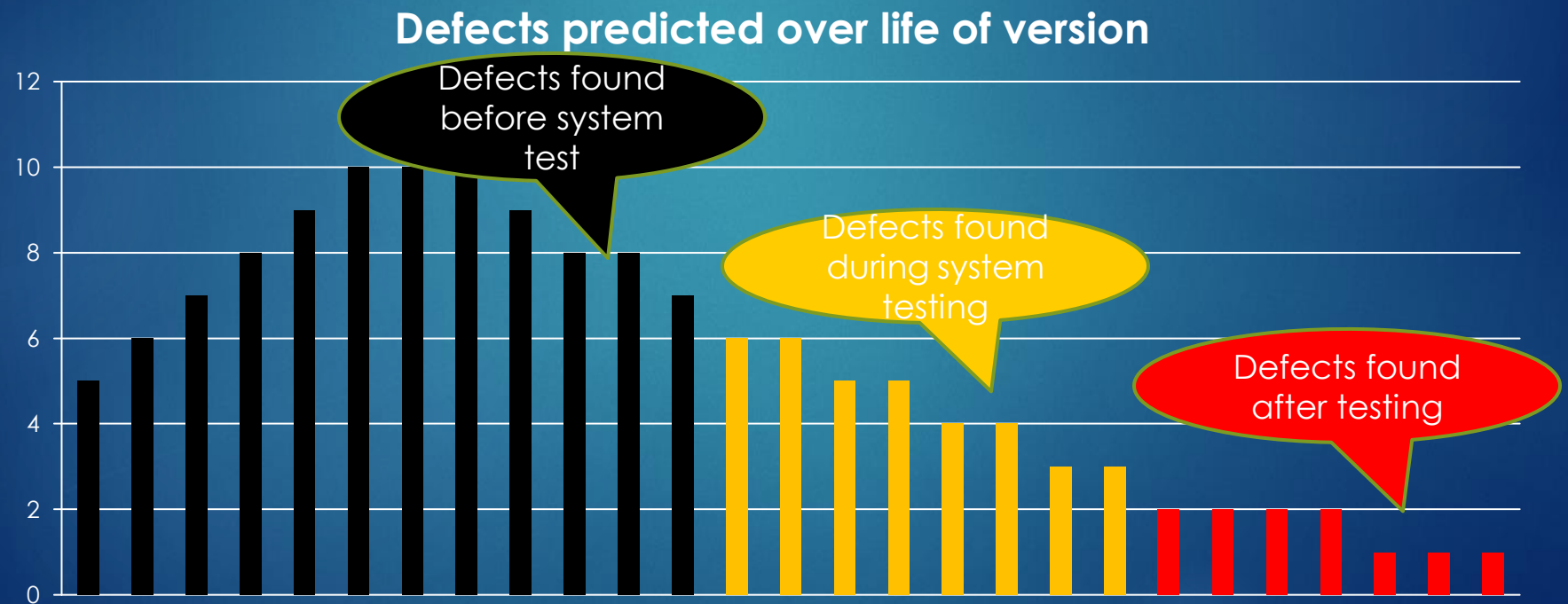
Prediction/ Assessment	Reliability Growth Estimations	Field reliability calculations
Used before code is written •Predictions can be incorporated into the system RBD •Supports planning •Supports sensitivity analysis •A few models have been available since 1987	**Used during system level testing or operation** •Determines when to stop testing •Validates prediction •Less useful than prediction for planning and avoiding problematic releases •Many models have been developed since 1970s such as the Musa Model. •The exponential model most commonly used.	**Used once software is operational** Actual failure rate, MTBF, etc is measured directly from field data

Software reliability assessment goals and outputs

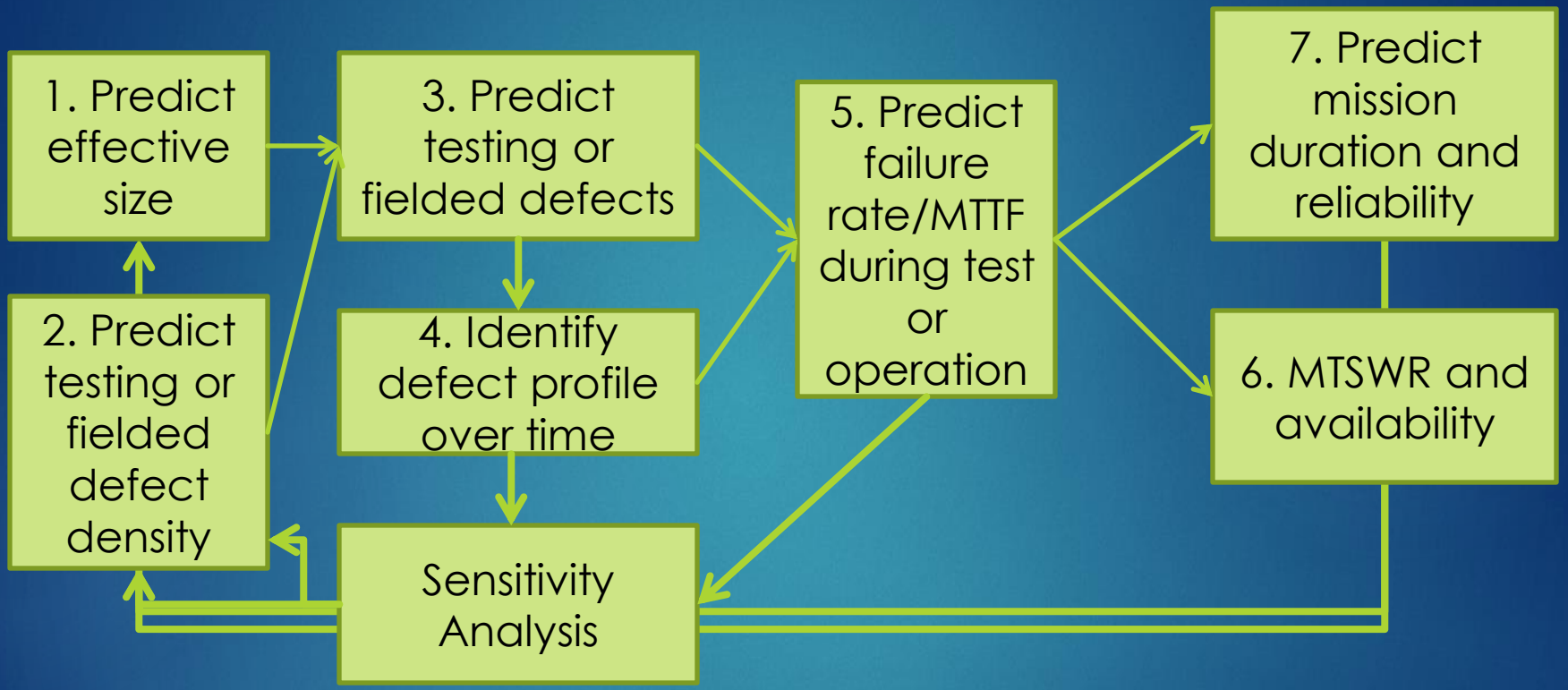
- Predict any of these reliability-related metrics
  - Defect density (test and operation)
  - Defects (test and operation)
  - Mean Time To Failure (MTTF), reliability, availability at any point in testing or operation
  - Reliability ty growth in any of the above metrics over time
  - Mean Time To Software Restore (MTSWR)
  - Maintenance and testing staffing levels to reach an objective
- Use prediction to
  - Analyze sensitivity to make a specific growth in one or more metrics
  - Analyze sensitivity between software and hardware
  - Benchmark defect density to others in the industry
  - *Identify practices that aren't effective in reducing defects*

If you can predict this defect profile you can predict failure rate

- For decades the defect profile has been the basis for nearly all software reliability models[2]
  - During development, you can predict the entire profile or parts of it
  - During testing, you can extrapolate the remainder of the profile



Industry approach to early software reliability predictions



1. Predict size

If everything else is equal, more code means more defects

- For in-house software
  - Predict the effective size of new, modified, and reused code using the best available industry method
- For COTS software (assuming the vendor can't provide effective size estimates)
  - Determine installed application size in KB (only EXEs and DLLs)
  - Convert application size to KSLOC using industry conversion
  - Assess reuse effectiveness by using a default multiplier of 1%
    - Accounts for the fact that COTS has been fielded to multiple sites

2. Available Methods for predicting defect density

- Ideally defect density prediction model optimizes simplicity, and accuracy and is updated on a regular basis

Method	Simplicity	Last updated on..	Accuracy
Predict defect density from historical data	Medium	N/A	Usually most accurate IF historical data is simple and recent
Predict defect density using an industry lookup chart or from SEI CMMi lookup chart	Easy	Varies	Usually the least accurate. Most useful for COTS software.
Predict defect density via surveys such as Shortcut, Full-scale, Rome Laboratory	Easy to Detailed	Softrel models are updated every 2 years Rome Labs model was last updated in 1992	If the survey is answered properly these are usually most accurate. RL model is geared only towards aircraft.

Survey Based Defect Density Models

Survey based model	Number of questions	Comments
Shortcut model	22	•More accurate than lookup charts •Questions can be answered by almost anyone familiar with the project
Rome Laboratory	45-212	•Some questions are outdated
Full-scale model A	98	•More accurate than the shortcut model •Questions require input from software leads, software testing, software designers
Full-scale model B	200	•More accurate than the Full-scale model A •Questions require input from software leads, software testing, software designers
Full-scale model C	300	•More accurate than the Full-scale model B •Questions require input from software leads, software testing, software designers •100 questions require expert review of development artifacts

How the Shortcut or Full-scale Survey Models Works

1. Complete survey and calculate score

Score

Predicted Percentile Group		Predicted Normalized *Fielded* Defect Density	Predicted Probability late delivery
1%	World class	.011	10%
10%	Very good	.060	20%
25%	Good	.112	25%
50%	Average	.205	36%
75%	Fair	.608	85%
90%	Poor	1.111	100%
99%	Distressed	2.069	100%

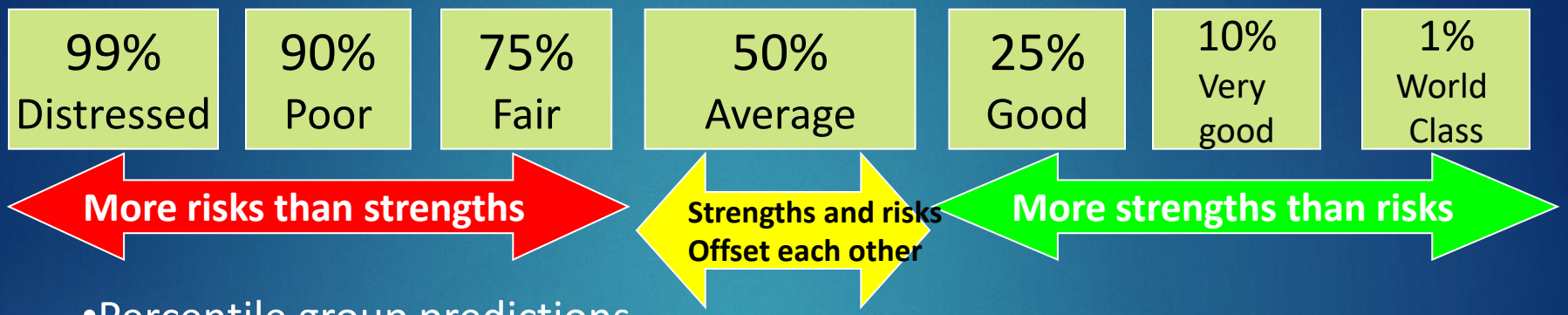
2.Find defect density and Probability (late) from associated row

3. When improving to next percentile

- Average defect reduction = 55%
- Average probability (late) reduction = 25%

Seven clusters used to predict defect density and ultimately software reliability

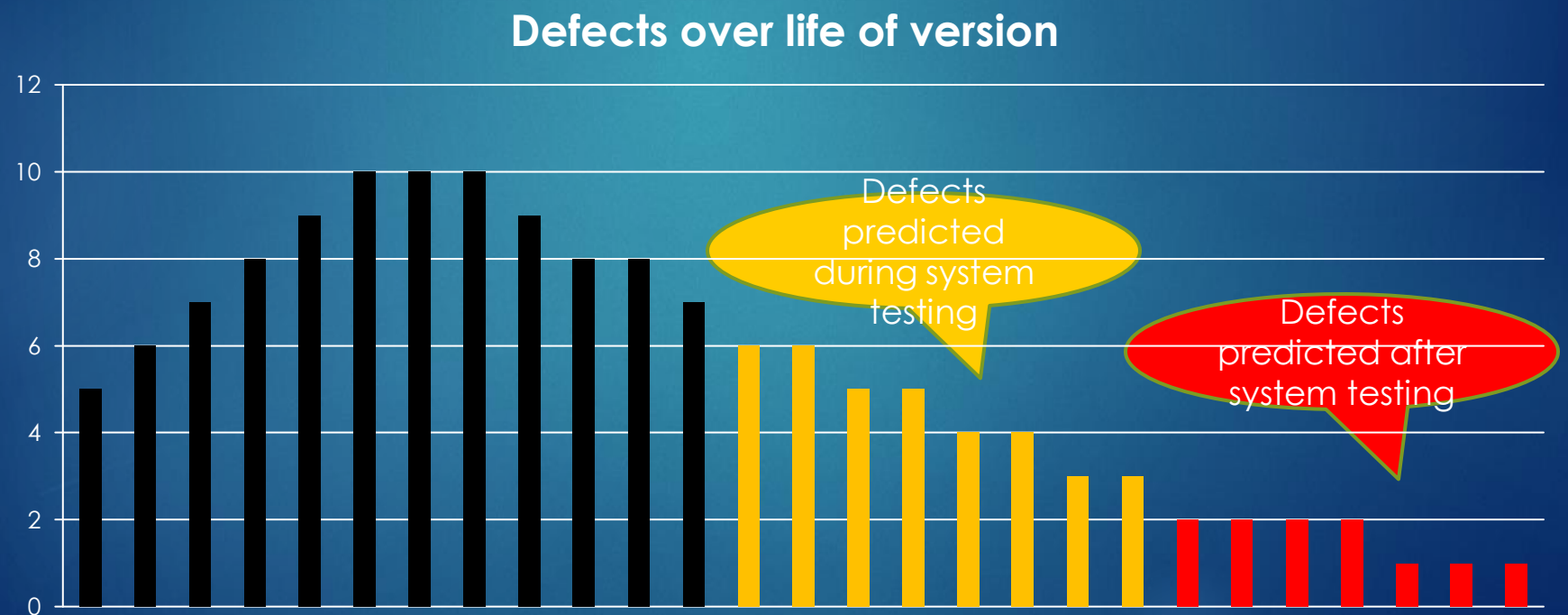
More fielded defects **Fewer fielded defects**



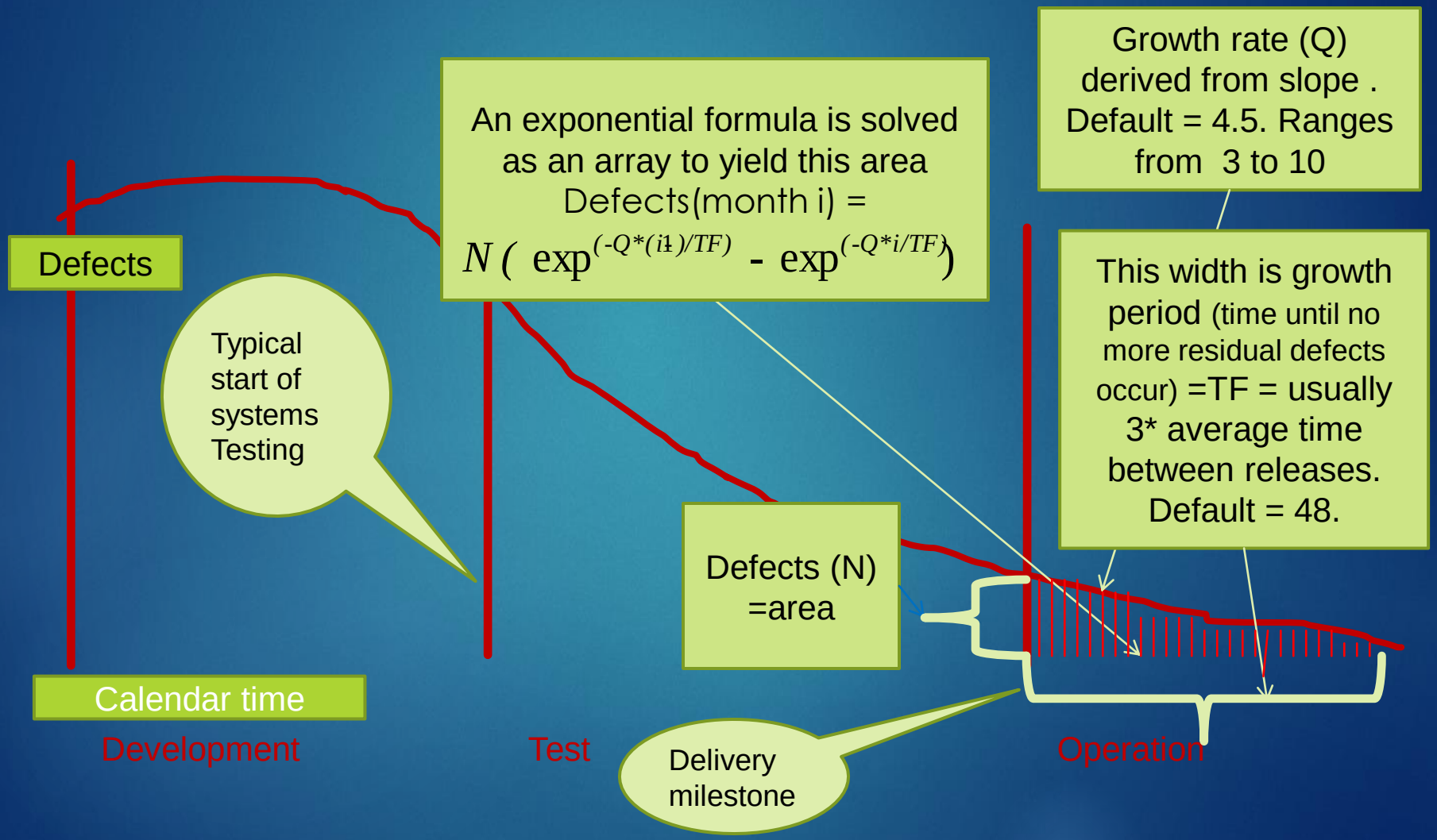
- Percentile group predictions…
  - Predicted directly from answering a survey and scoring it
  - Pertain to a particular product version
  - Can only change if or when risks or strengths change
  - Some risks/strengths are temporary; others can't be changed at all
  - Can transition in the wrong direction on same product if
    - New risks/obstacles added
    - Opportunities are abandoned
  - World class does not mean defect free. It simply means better than the defect density ranges in database.

3. Predict testing or fielded defects

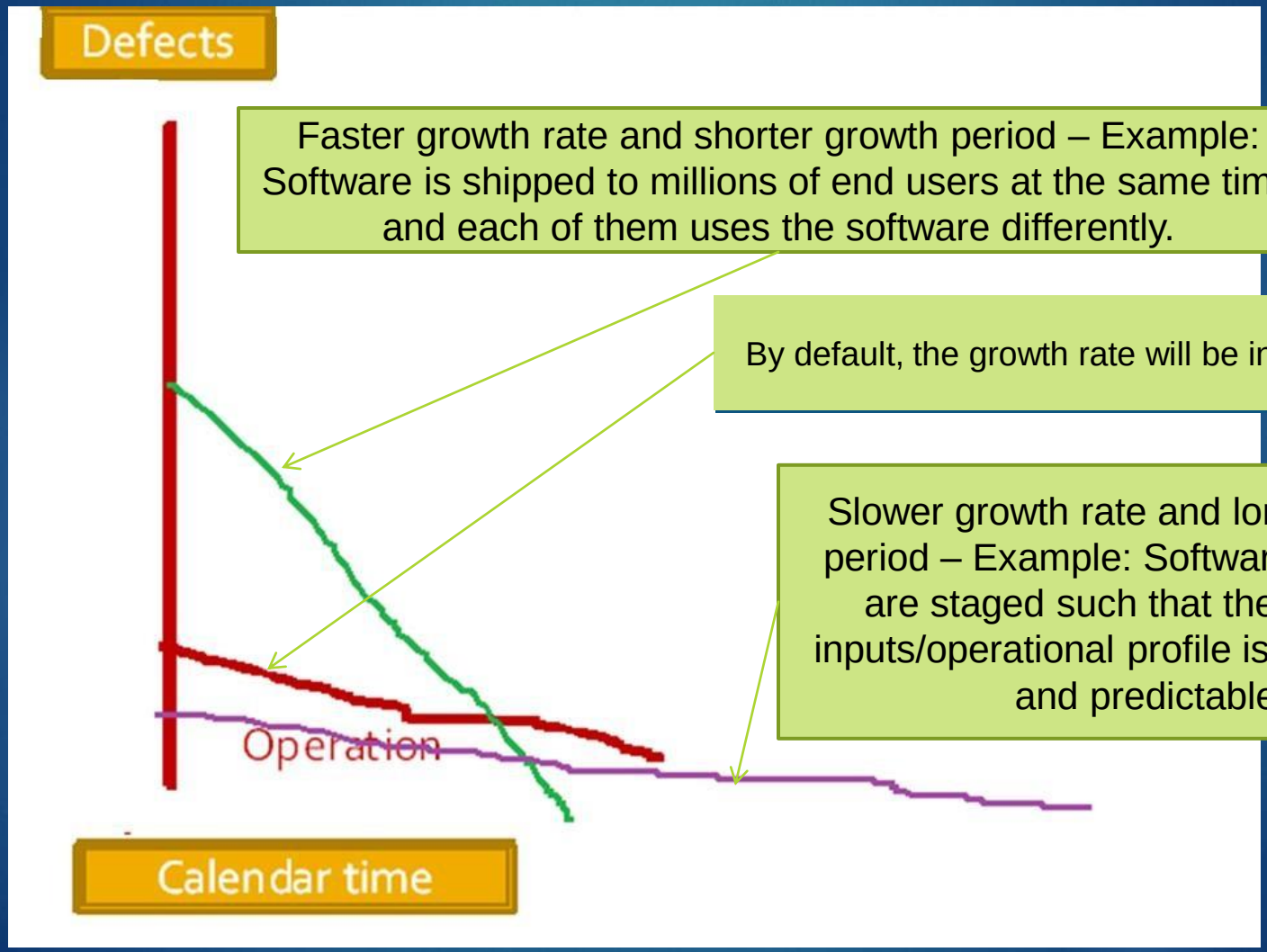
- Defects can be predicted as follows
  - Testing defect density * Effective size = Defects predicted to be found during testing (Entire yellow area)
  - Fielded defect density * Effective size = Defects predicted to be found in operation (Entire red area)



4. Identify shape of defect profile



Rate at which defects result in observed failures (growth rate)



5. Use defect profile to predict failure rate/MTTF

- Dividing the defect profile by the duty cycle profile yields a prediction of the failure rate as shown next
- T_i= duty cycle for a month i - how much the software is operated during some period of calendar time. Ex:
  - If the software is operating 24/7 ->duty cycle is 730 hours per month
  - If software operates during normal working hours ->duty cycle is 176 hours per month
- MTTF $_i$= $\frac{T_i}{Defectprofile_i}$
- MTTCF $_i$ $\frac{T_i}{\%severe * Defectprofile_i}$
  - % severe = % of all fielded defects that are predicted to impact availability

6. Predict MTSWR (Mean Time To Software Restore) and Availability

- Needed to predict availability
- For hardware, MTTR is used. For software, MTSWR is used.
- MTSWR =weighted average of time for applicable restore actions by the expected number of defects that are associated with each restore action
- Availability profile over growth period = Availability_i=

$$\frac{MTTCF_i}{MTTCF_i + MTSWR}$$

- In the below example, MTSWR is a weighted average of the two rows

Operational restore action	Average restore time	Percentage weight
Correct the software	40 hours	.01
Restart or reboot	15 minutes	.99

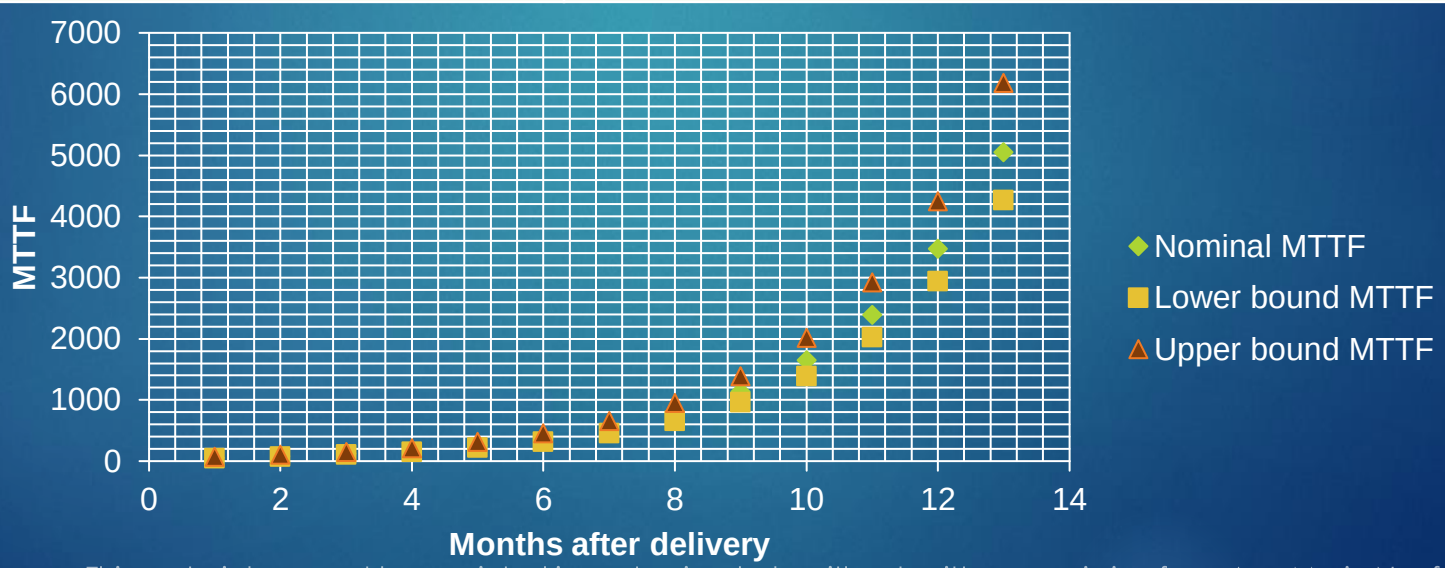
7. Predict mission time and reliability

- Reliability profile over-growth period =
  - $R_i = \exp(-\text{mission time} / \text{MTTCF}_i)$
- Mission time = how long the software will take to perform a specific operation or mission
  - Not to be confused with duty cycle or testing time
  - Example: A typical dishwasher cycle is 45 minutes. The software is not executing outside of this time, so reliability is computed for the 45-minute cycle.

Confidence Bounds and prediction error

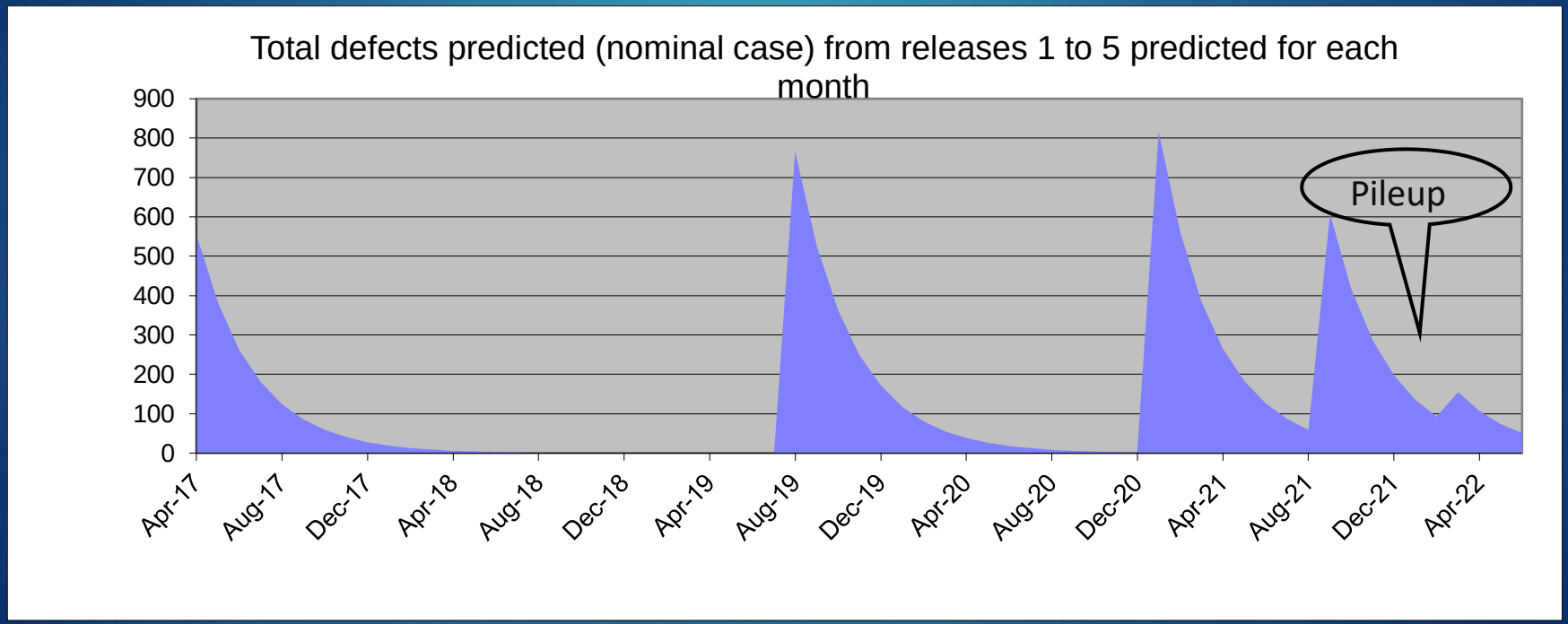
- Software prediction confidence bounds are a function of

Parameter	Contribution to prediction error
Size prediction error due to scope change	Until code is complete, this will usually have the largest relative error
Size prediction error due to error in sizing estimate (scope unchanged)	Minimized with use of tools, historical data
Defect density prediction error	Minimized by validating model inputs
Growth rate error	Not usually a large source of error



Predictions can be used for scheduling and maintenance

- Predictions can be used to determine how far apart releases should be to optimize warranty costs and response time
- This is an example from industry. The defects were predicted to pileup up after the third release.



Sensitivity analysis and defect reduction

- SoftRel survey models and the Rome Laboratory model were developed to support defect reduction scenario analysis
- Use the models to find the gaps and determine the sensitivity of each gap
- Develop strategies for reducing the defects and rework the predictions based on a few key improvements

Know which software characteristics/practices have biggest impact on software reliability

- To date, 600+ characteristics related to the 3 P's have been mathematically correlated to software reliability by SoftRel, LLC[1]
  - Product/industry/application type
  - People
  - Practices/process
- Of these, 120 are so strongly related that they are used collectively to predict before the code is even written

[1]See the entire research and complete list of practices at "The Cold Hard Truth About Reliable Software", A. Neufelder, SoftRel, LLC, 2014

Research results revealed some surprises

- Some practices, tools, and metrics don't always result in better software when…
  - Required prerequisites may not be in place
  - Required training may not be in place
  - Practices, tools, or metrics used incorrectly
  - Software groups not mature enough to implement practice, tool, or metric
  - Metric provides results that aren't useful

Practice that's not always related to lower defect density	**Why**
Expensive automated design and testing tools	Requires training and maturity
Peer code reviews	Agenda is often adhoc or superficial
Complexity and depth of nesting metrics	Correlated at extremely high or low values but not in between
Advanced software life cycle models	Model not executed properly or it's not the right model for this software product

These are the 10 factors mostly strongly related to software reliability

1. Software engineers have product/industry domain expertise
2. Do formal white/clear box unit testing
3. Start writing test plans before any code is written
4. Outsource features that aren't in your organization's line of business
5. Avoid outsourcing features that are your organization's line of business
6. Don't skip requirements, design, unit test, or system testing even for small releases
7. Plan ahead – even for small releases. Most projects are late because of unscheduled defect fixes from the previous release (and didn't plan on it)
8. Reduce "Big Blobs" - big teams, long milestones - *especially* when you have a large project
9. Don't use automated tools until the group has expertise in whatever the tool is automating
10. Define in writing what the software should NOT do

Conclusions

- Software reliability can be predicted before the code is written
  - It can be applied to COTS software as well as custom software
  - A variety of metrics can be predicted
  - The predictions can be used for sensitivity analysis and defect reduction
  - A variety of methods exist depending on how much data is available

Frequently Asked Questions

- Can I predict the software reliability when there is an agile or incremental software development lifecycle?
  - Yes, your options are
    - You can use the models for each internal increment and then combine the results of each internal increment to yield a prediction for each field release
    - You can add up the code size predicted for each increment and do a prediction for the field release based on the sum of all increment sizes
- How often are the predictions updated during development?
  - Whenever the size estimates have a major change or whenever there is a major review
  - The surveys are not updated once complete unless it is known that something on the survey has changed
    - i.e. there is a major change in staffing, tools, or other resource during development, etc.

Frequently Asked Questions

- Which defect density prediction models are preferred?
  - The ones that you can complete accurately and the ones that reflect your application type
    - If you can't answer most of the questions in a particular mode survey then you shouldn't use that model
    - If the application lookup charts don't have your application type you shouldn't use them
- How can I get the defect density prediction models?
  - Software Reliability Toolkit Training Class
  - Software Reliability Toolkit
  - Frestimate Software

References

- [1] "The Cold Hard Truth About Reliable Software", A. Neufelder, SoftRel, LLC, 2014
- [2]Four references are

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) Section 8 of MIL-HDBK-338B, 1 October 1998

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

Related Terms

- Error
  - Related to human mistakes made while developing the software
  - Ex: Human forgets that b may approach 0 in algorithm c = a/b
- Fault or defect
  - Related to the design or code
  - Ex: This code is implemented without exception handling "c = a/b;"
  - Defect rate is from the developer's perspective
  - Defects measured/predicted during testing or operation
  - Defect density = defects/normalized size
- Failure
  - An event
  - Ex: During execution, the conditions are so that the value of b approaches 0 and the software crashes or hangs
  - The failure rate is from the system or end user's perspective
- KSLOC
  - 1000 source lines of code – the common measure of software size