Predict Software Reliability Before the Code is Written

Ann Marie Neufelder
SoftRel, LLC
ann.neufelder@missionreadysoftware.com
http://www.missionreadysoftware.com

Copyright

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 applications

IEEE 1633 Rewritten to be practical

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

Prediction/ Assessment	Reliability Growth Models
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.
Section of IEEE 1633 Recommended Practices for Software Reliability, 2016	
5.3	5.4

Limitations of each type of modeling

PREDICTION/ASSESSMENT

- All are based on historical actual data
- All generate a prediction by calibrating current project against historical project(s)
- Accuracy depends on
  - How similar historical data is to current project
    - Application type
    - Product stability (version 1 versus version 50)
    - Capabilities of the development team
  - How current the historical data is
  - How much historical data exists

RELIABILITY GROWTH MODELS

- All are based on extrapolating an existing trend into the future
- Accuracy depends on
  - Test coverage
    - Low test coverage usually results in optimistic results
  - How closely actual trend matches assumed trend
    - i.e. if model assumes a logarithmic trend is that the actual trend?
  - How closely the model assumptions match actual
    - Defect removal
    - Defect independence

PREDICTIONS/ASSESSMENTS

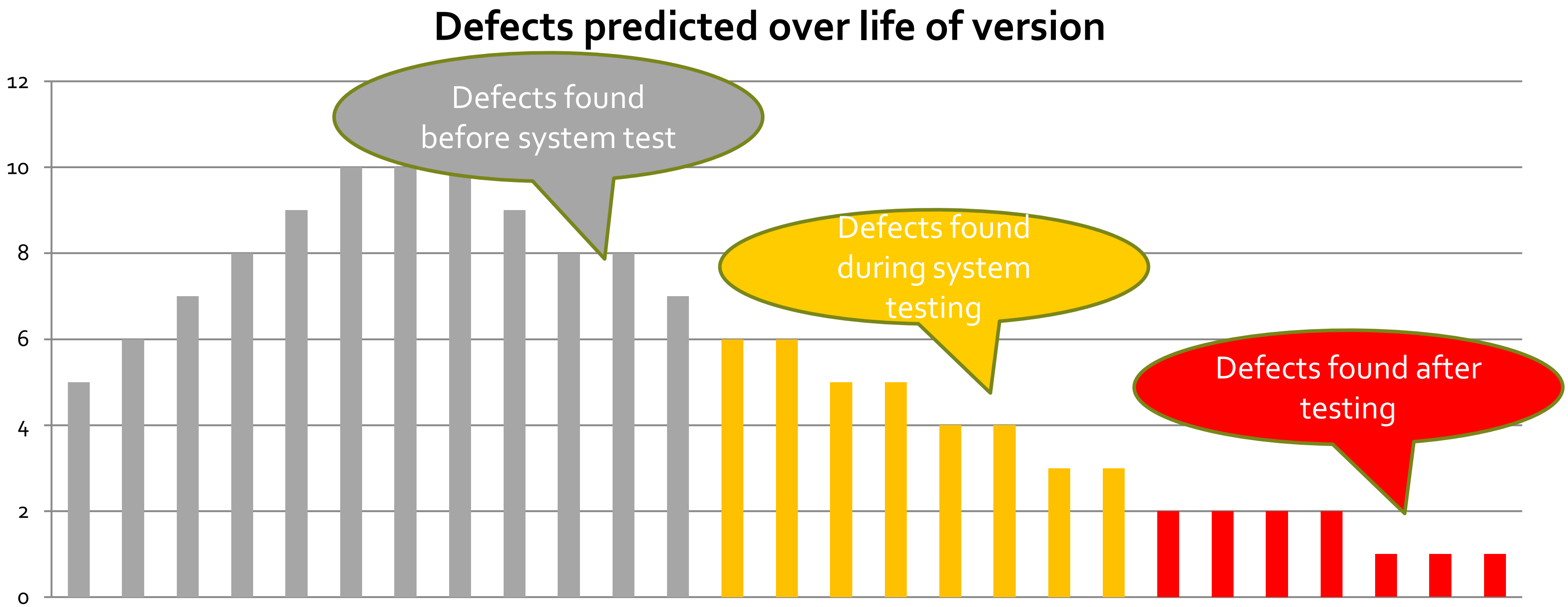
Overview

Software reliability assessment goals and outputs

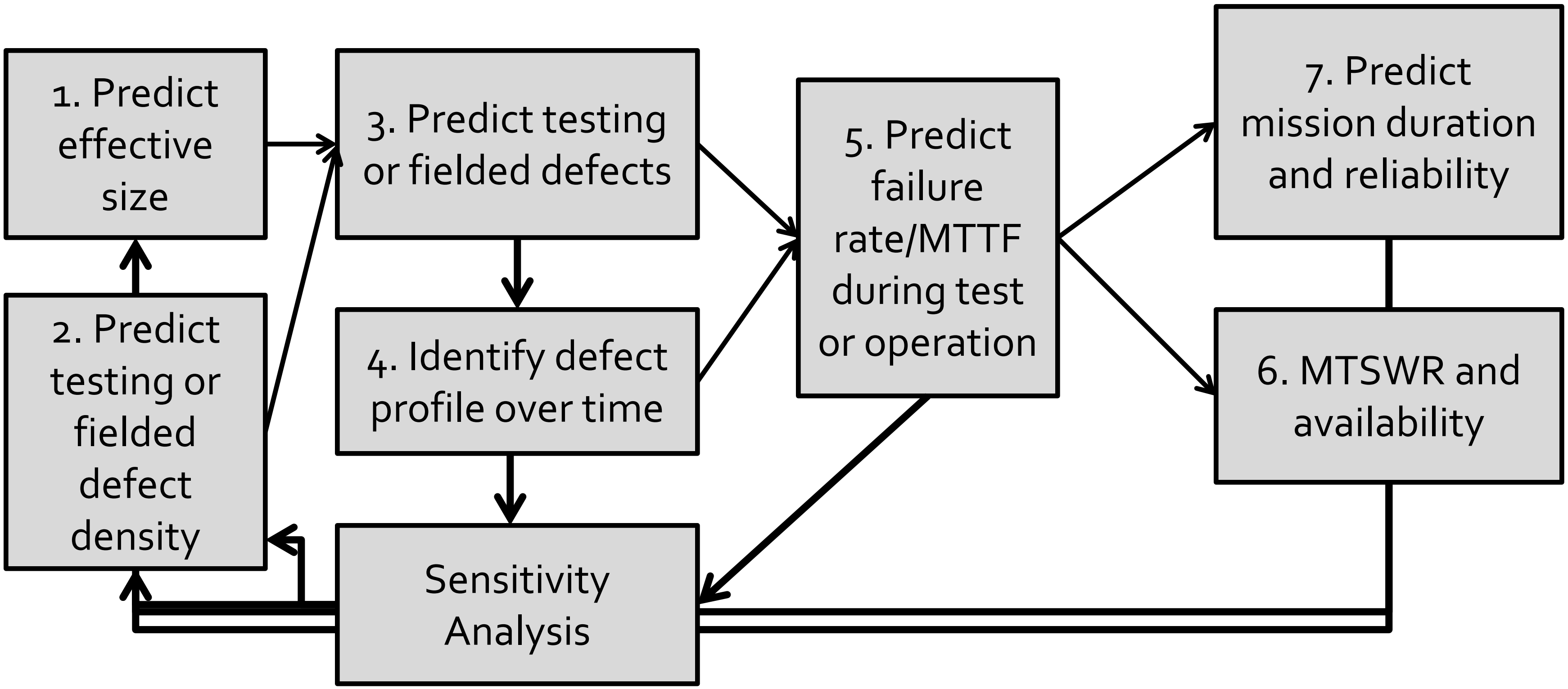
- Predict any of these reliability related metrics
  - Defect density (test and operation)
  - Defects (test and operation)
  - Mean Time To Failure (MTTF), reliability, availability at any point in testing or operation
  - Reliability ty growth in any of the above metrics over time
  - Mean Time To 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 industry
  - *Identify practices that aren't effective for 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 framework for early software reliability predictions



This framework has been used for decades. What has changed over the years are the models for steps 1, 2 and 4. These models evolve because software languages, development methods and deployment life cycles have evolved.

If everything else is equal, more code means more defects

- For in house software
  - Predict effective size of new, modified and reused code using best available industry method
- For COTS software (assuming 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 default multiplier of 1%
    - Accounts for 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 asessments such as Shortcut, Full-scale, Rome Laboratory, Neufelder models.	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.

* These models are recommended in the normative section of the IEEE 1633 Recommended Practices for Software Reliability, 2016.

Assessment 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
Neufelder model	149	•Based on Process Grade Factors

* These models are recommended in the normative section of the IEEE 1633 Recommended Practices for Software Reliability, 2016. ** These models are recommended in Annexes of IEEE 1633 Recommended Practices for Software Reliability, 2016.

How the Assessment Models Works

1. Complete assessment and calculate score

Score

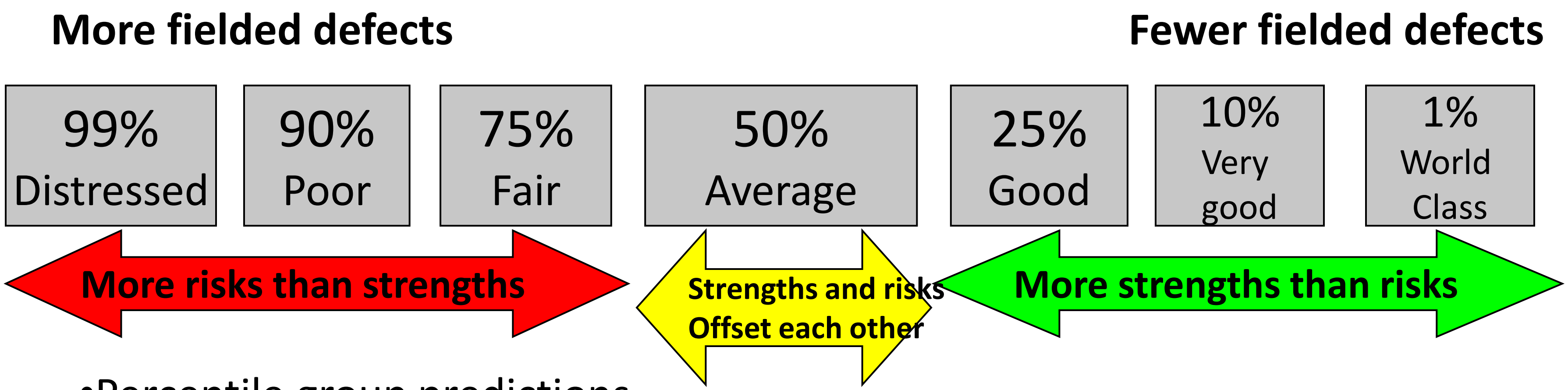
Predicted Percentile Group		Predicted Normalized *Fielded* Defect Density	Predicted Probability late delivery
1%	World class	.011	10%
10%	Very good	.060	20%
25%	Good	.112	25%
50%	Average	.205	36%
75%	Fair	.608	85%
90%	Poor	1.111	100%
99%	Distressed	2.069	100%

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

3. When improving to next group

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

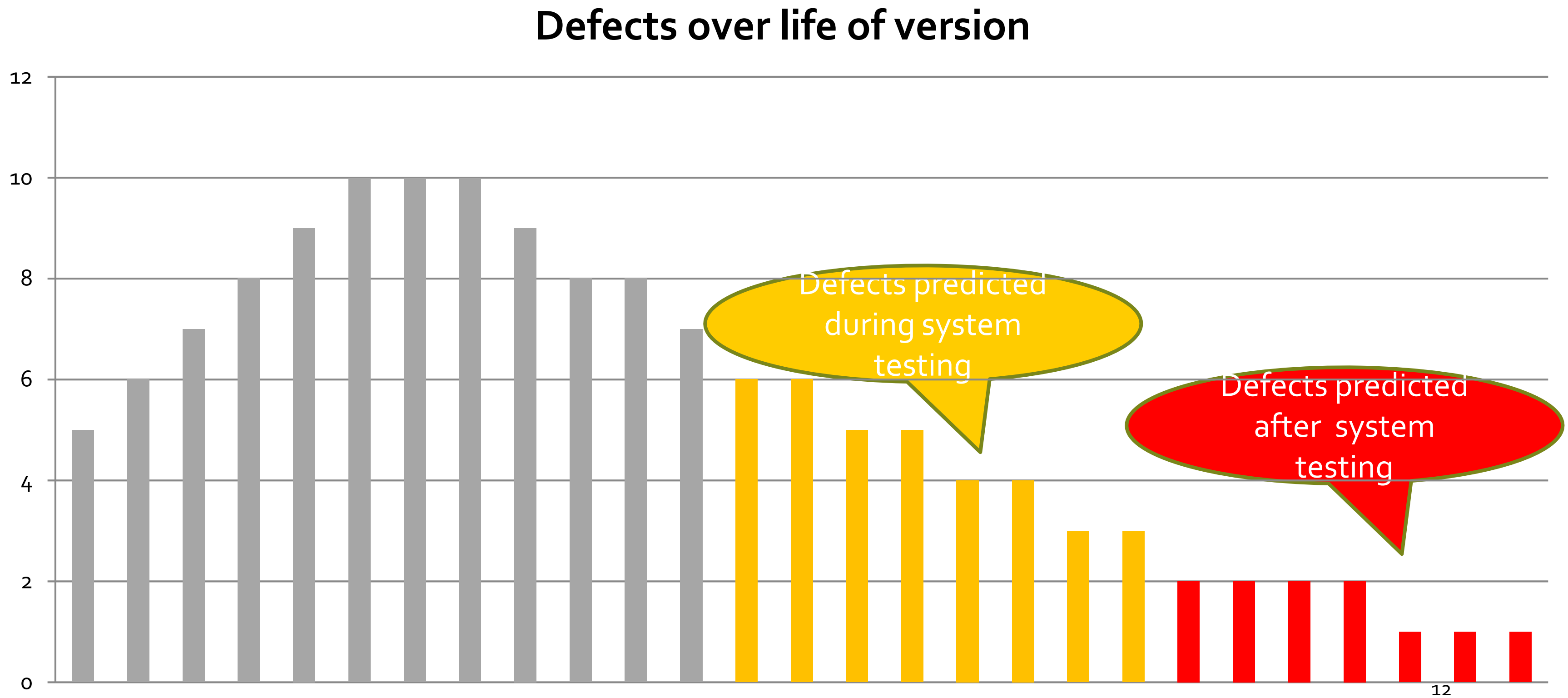
Seven clusters used to predict defect density and ultimately software reliability



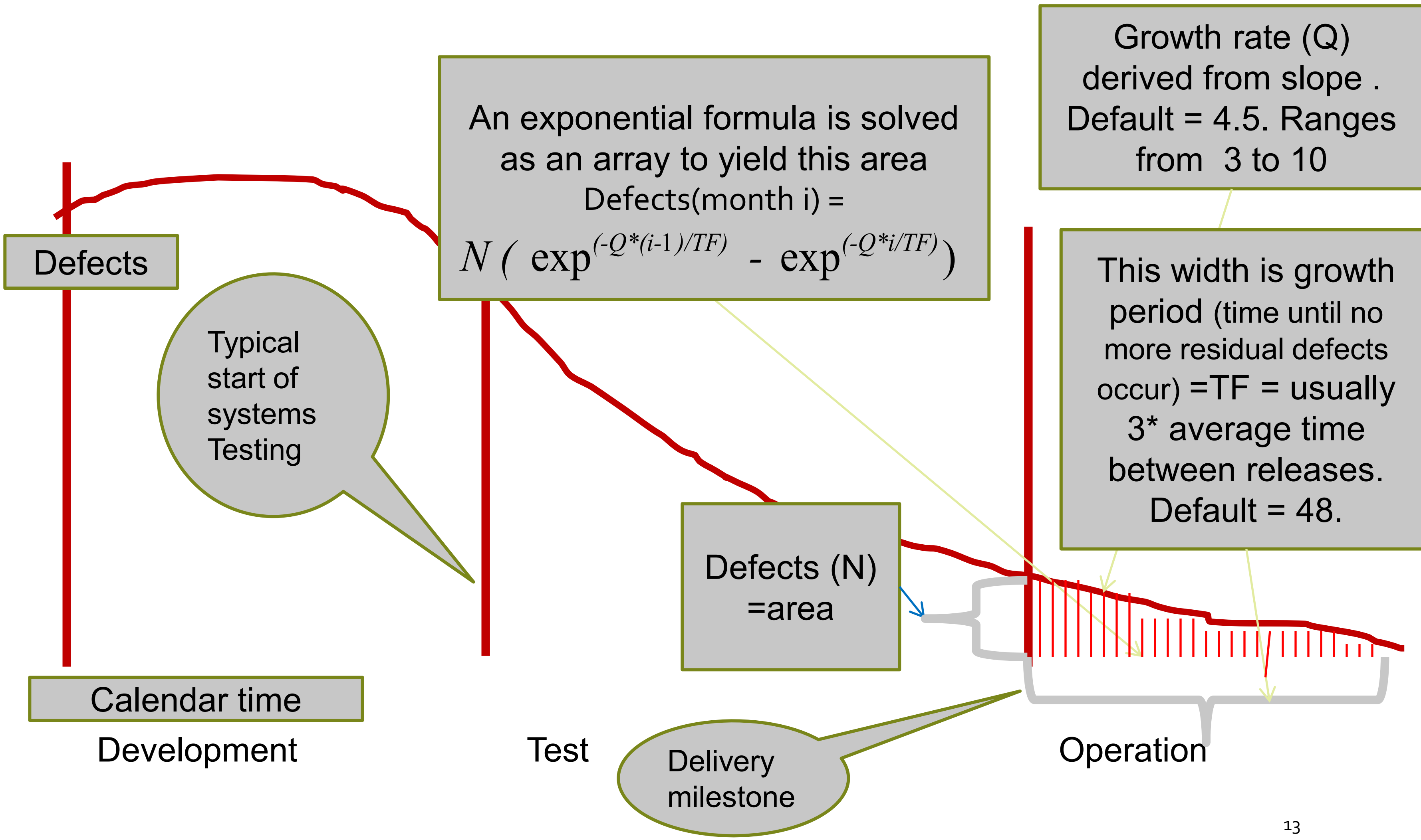
- Percentile group predictions…
  - Predicted directly from answering a survey and scoring it
  - Pertain to a particular product version
  - Can only change if or when risks or strengths change
  - Some risks/strengths are temporary; others can't be changed at all
  - Can transition in the wrong direction 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

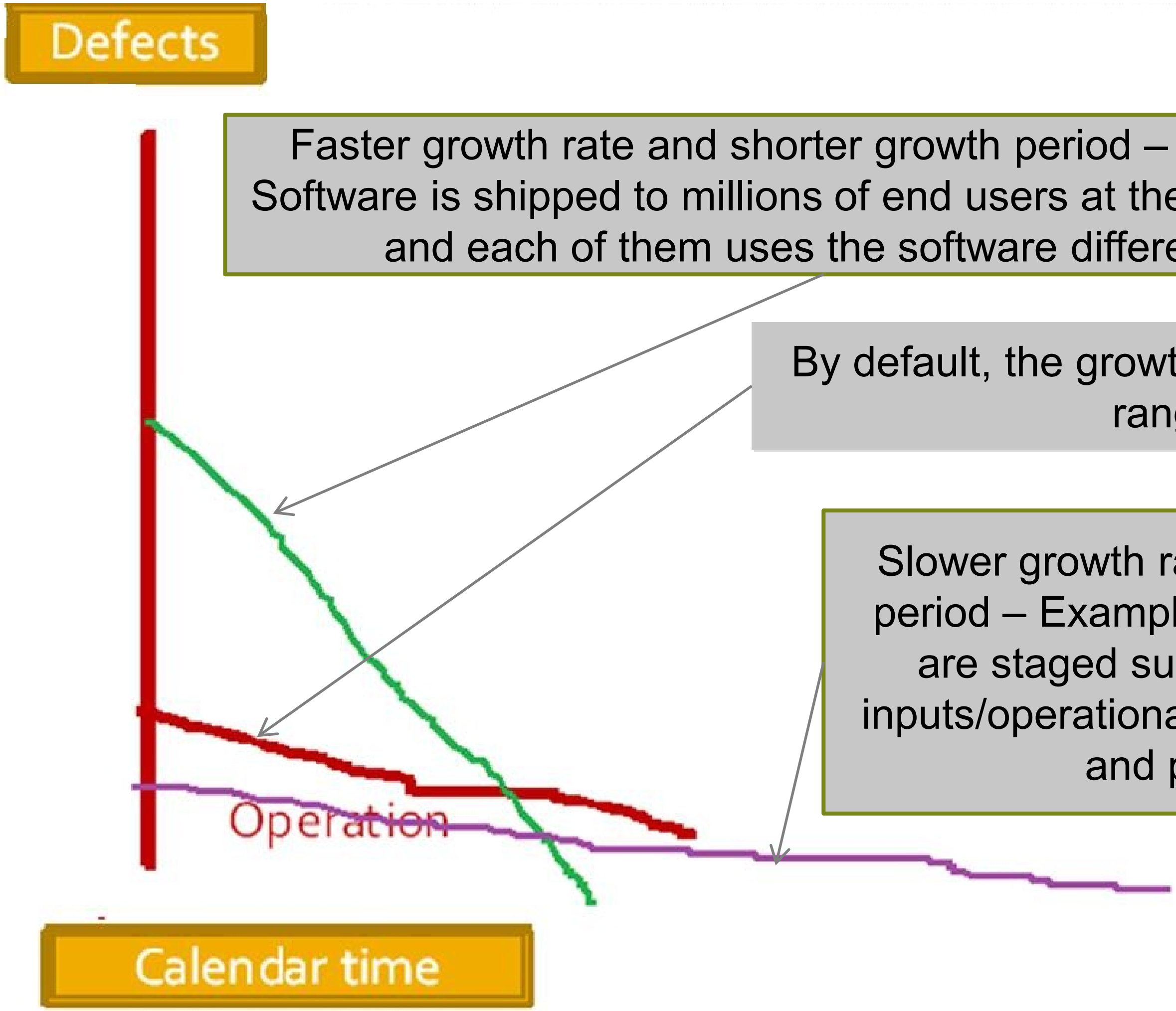
- Defects can be predicted as follows
  - Testing defect density * Effective size = Defects predicted to be found during testing (Entire yellow area)
  - Fielded defect density * Effective size = Defects predicted to be found in operation (Entire red area)



4. Identify shape of defect discovery profile



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



Faster growth rate and shorter growth period – Example: Software is shipped to millions of end users at the same time and each of them uses the software differently.

By default, the growth rate will be in this range

Slower growth rate and longer growth period – Example: Software deliveries are staged such that the possible inputs/operational profile is constrained and predictable

5. Use defect discovery profile to predict failure rate/MTTF

- Dividing defect profile by duty cycle profile yields a prediction of failure rate as shown next
- T_i= duty cycle for month i - how much the software is operated during some period of calendar time. Ex:
  - If 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 $_i$
- % severe = % of all fielded defects that are predicted to impact availability

$$\frac{T_i}{\%severe * Defectprofile_i}$$

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 =
  - $R_i = \exp(-^{\text{mission time}/\ MTTCF}{}_i)$
- 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.

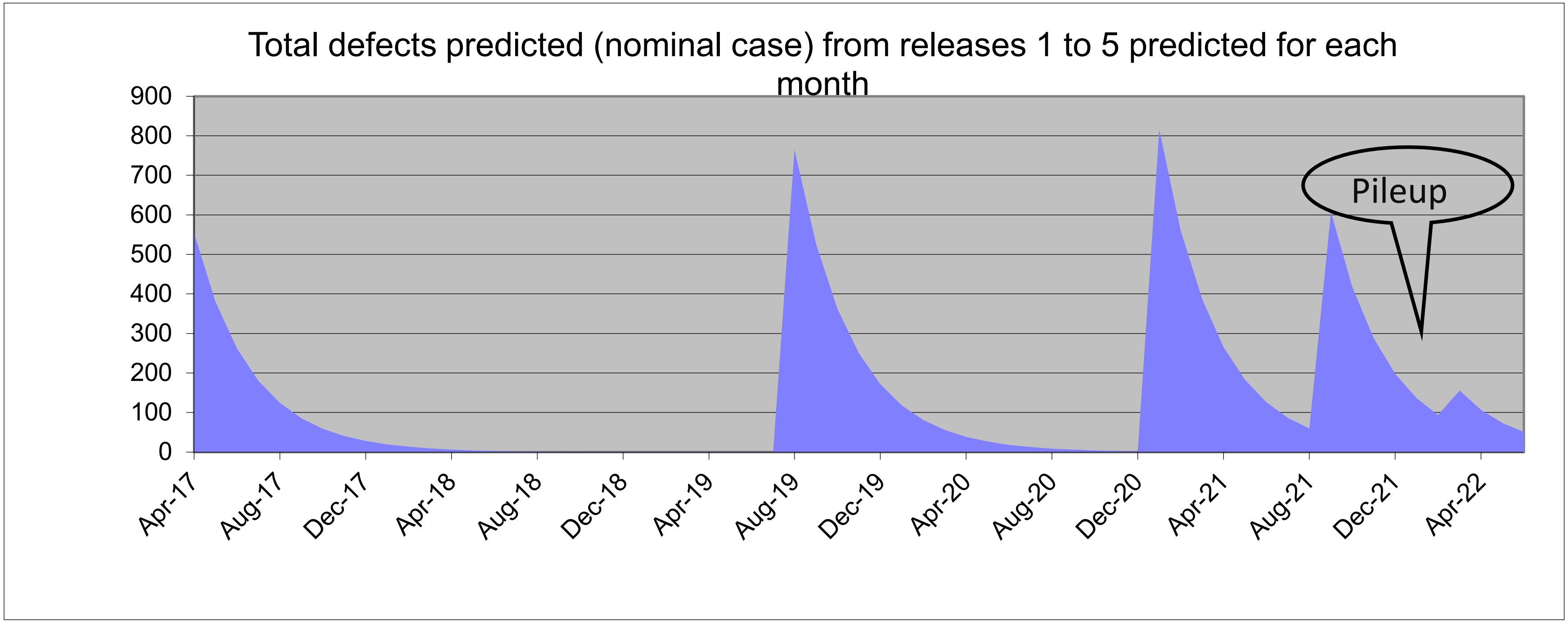
Confidence Bounds and prediction error

- 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

- Assessment models were developed for **the** purpose of supporting defect reduction scenario analysis
- Use the models to find the gaps and determine 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, 2019

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
Advanced software life cycle models	Model not executed properly or it's not the right model for this software product

Research results revealed some surprises

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

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 group has expertise in whatever the tool is automating
10. Define in writing what the software should NOT do

These are the 10 factors mostly strongly related to software reliability

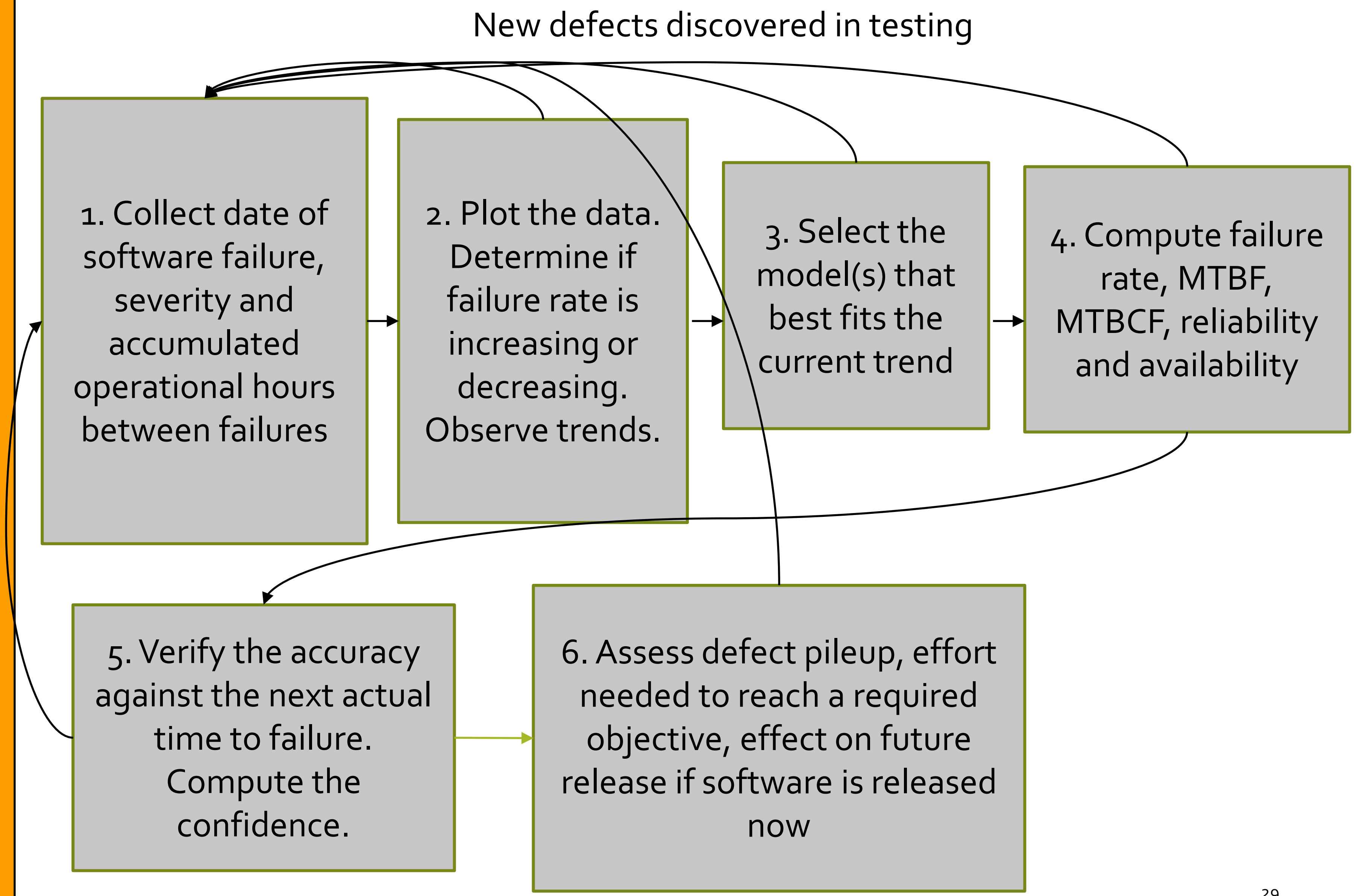
RELIABILITY GROWTH MODELS USED DURING TESTING

Overview

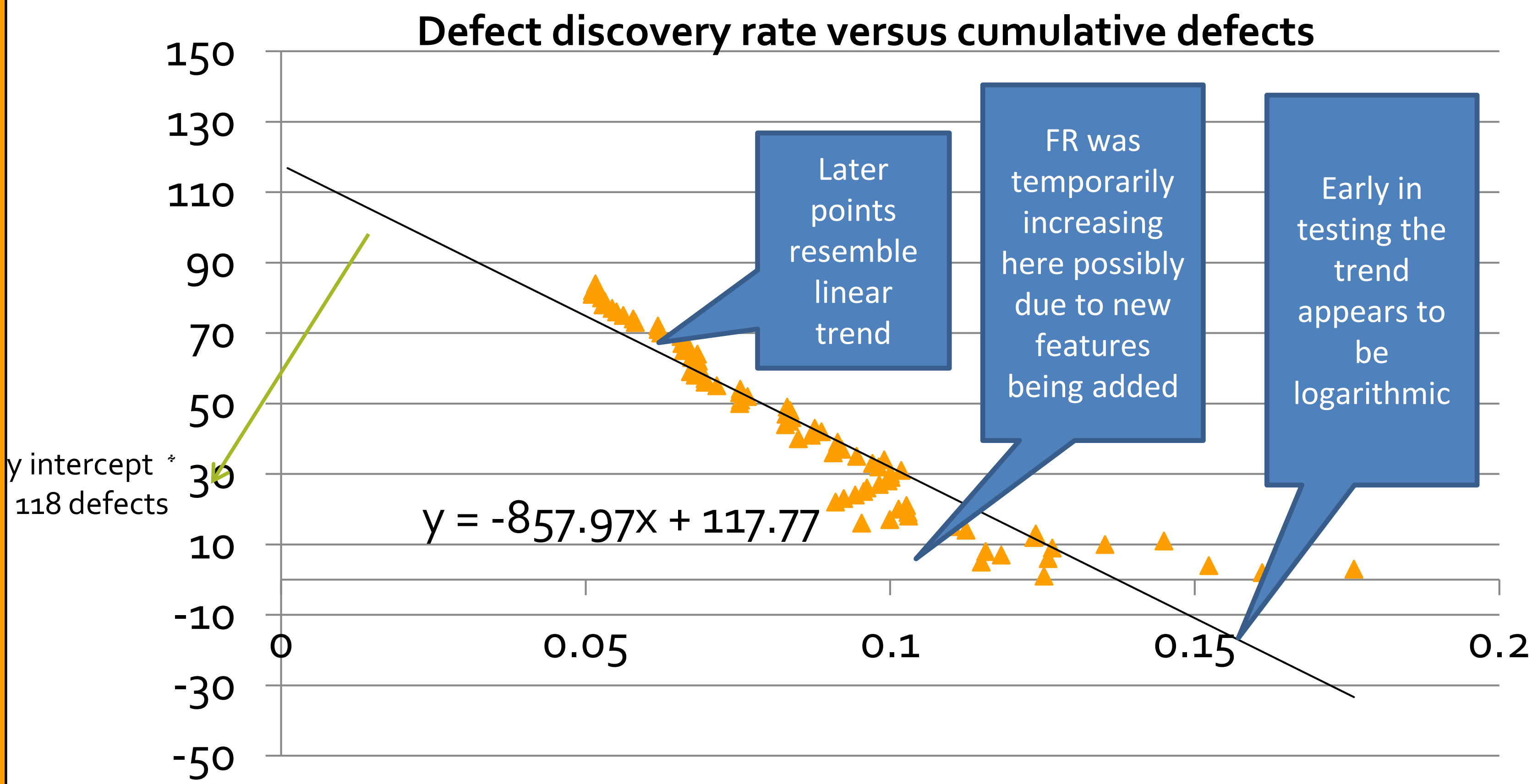
- Reliability growth models have been in use since the 1970s for software reliability
- Due to exceedingly poor documentation and guidance by Academic community, there has been unnecessary confusion regarding how to use the models
- This was resolved in the 2016 edition of the IEEE Recommended Practices for Software Reliability.
  - Overview of the models
  - How to select the model(s)
  - When to use them and when not to
  - How to use with incremental development life cycle

Overview

Reliability Growth Model framework

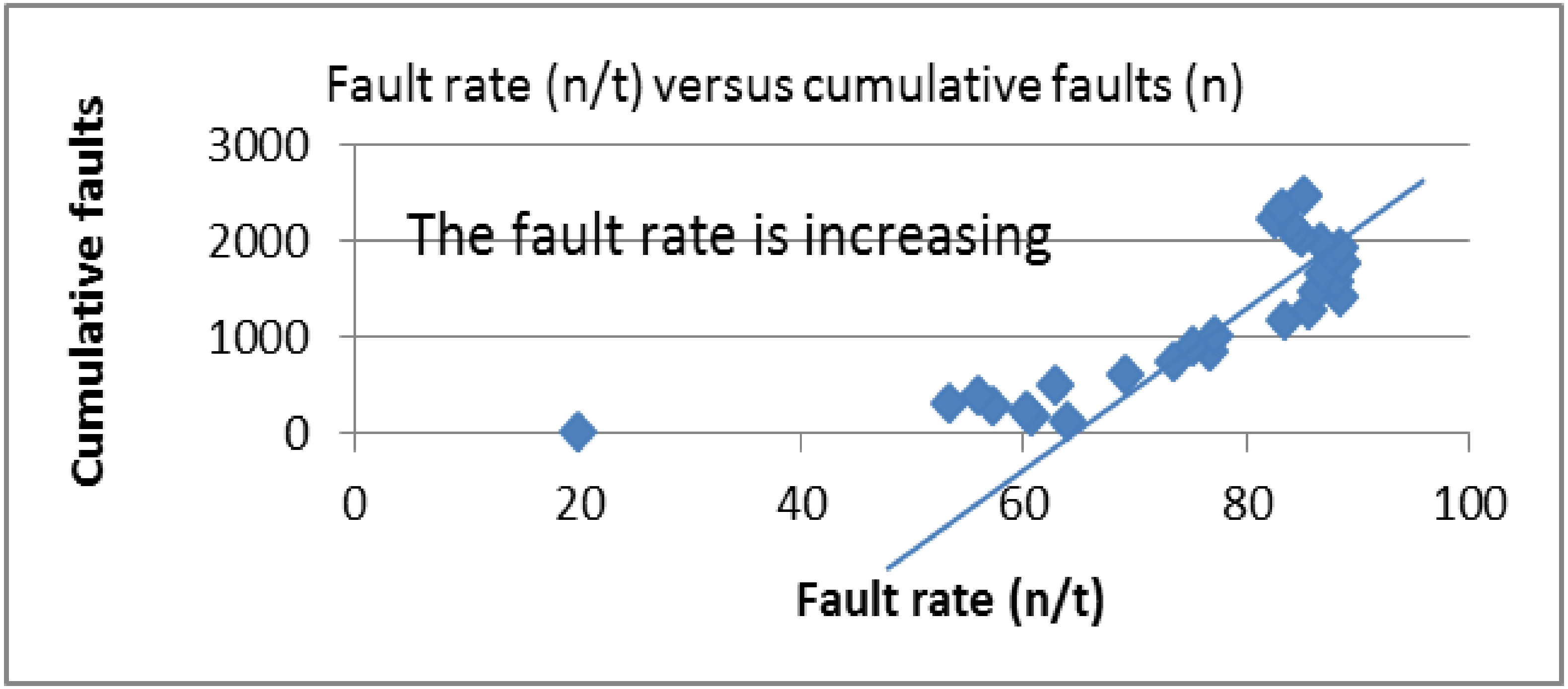


Example of defect discovery data plot



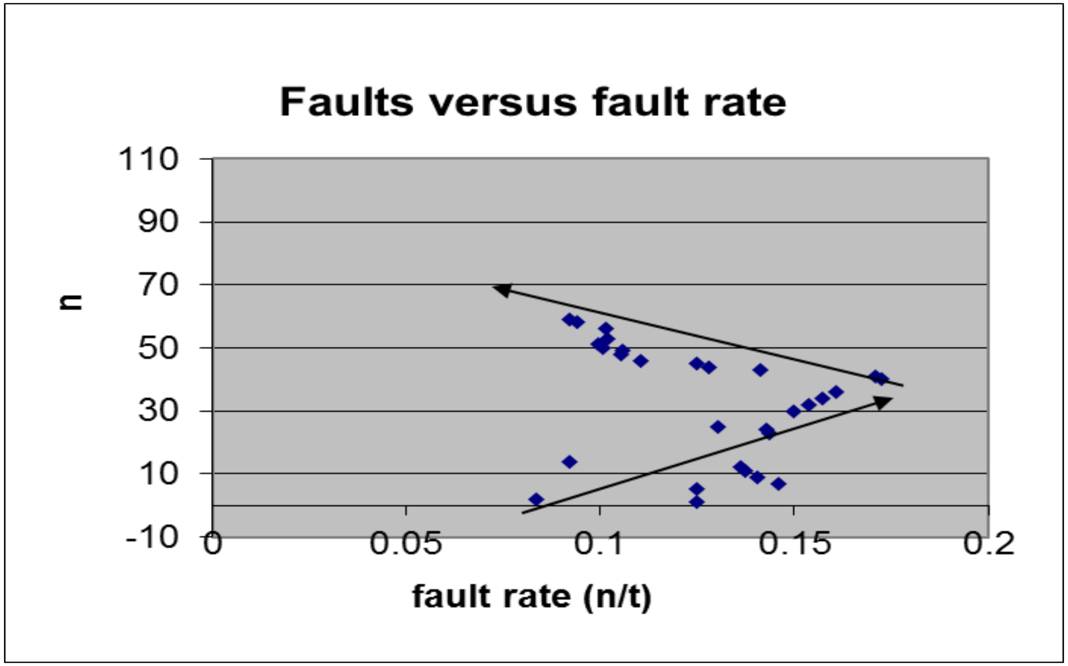
- In this example, the defect discovery rate is generally decreasing. There was one point during testing in which it temporarily was increasing. This is why data needs to be collected regularly and plotted regularly.

Example of defect discovery data plot



- In this example, the defect discovery rate (fault rate) is increasing. This means that only a few models can be used.

Example of defect discovery data plot



- In this example, the defect discovery rate increased initially and then decreased steadily. In this case the most recent data can be used to extrapolate the future trend.

Selecting the reliability growth model(s)

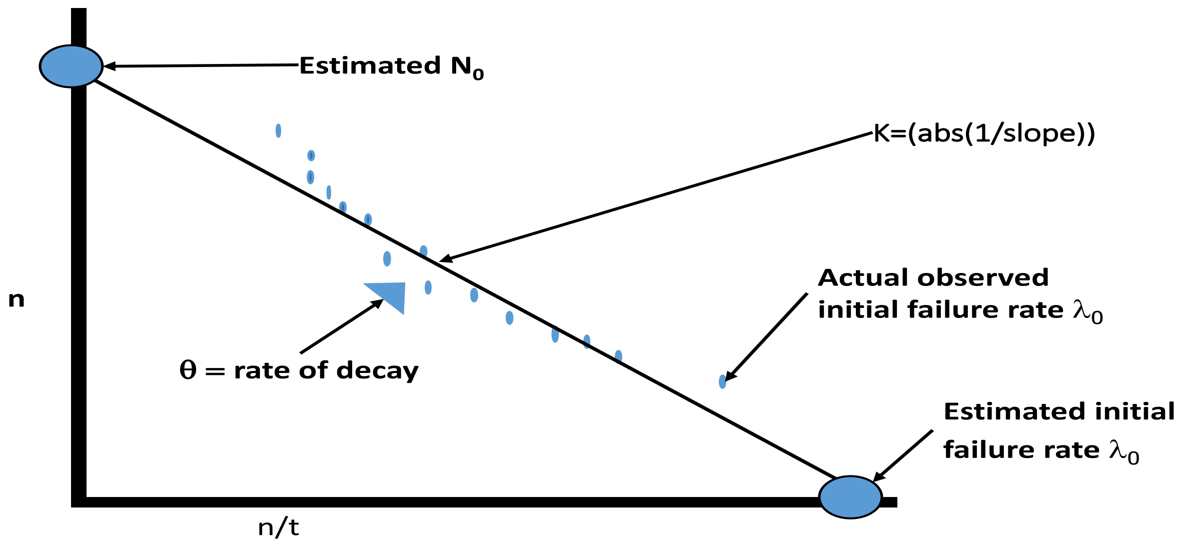
Model name	Inherent defect count	Effort required (1 low, 3 high)	Can be used when exact time of failure unknown
Increasing fault rate			
Weibull [B46]	Finite/not fixed	3	Yes
Peaked fault rate			
Shooman Constant Defect Removal Rate Model[B61]	**Finite/fixed**	**1**	**Yes**
Decreasing fault rate			
Shooman Constant Defect Removal Rate Model[B61]	**Finite/fixed**	**1**	**Yes**
Linearly Decreasing			
General exponential models including: • Goel-Okumoto [B47] • Musa Basic Model[B45] • Jelinski-Moranda [B48]	Finite/fixed	2	Yes
Shooman Linearly Decreasing Model[B61]	**Finite/fixed**	**1**	**Yes**
Non-Linearly Decreasing			
Musa-Okumoto (logarithmic) [B50]	**Infinite**	**1**	**Yes**
Shooman Exponentially Decreasing Model[B62]	Finite/fixed	3	Yes
Log-logistic [B51]	Finite/fixed	3	Yes
Geometric [B52]	Infinite	3	No
Increasing and then decreasing			
Yamada (Delayed) S-shaped [B53]	Infinite	3	Yes
Weibull [B46]	Finite/not fixed	3	Yes

1. Eliminate models that don't fit the observed trend.
2. Use all applicable models or select the one with least effort.
3. Some models expect exact time of failure which might not be easy to collect in testing.

Bolded models are in normative section of IEEE 1633 Recommended Practices for Software Reliability, 2016

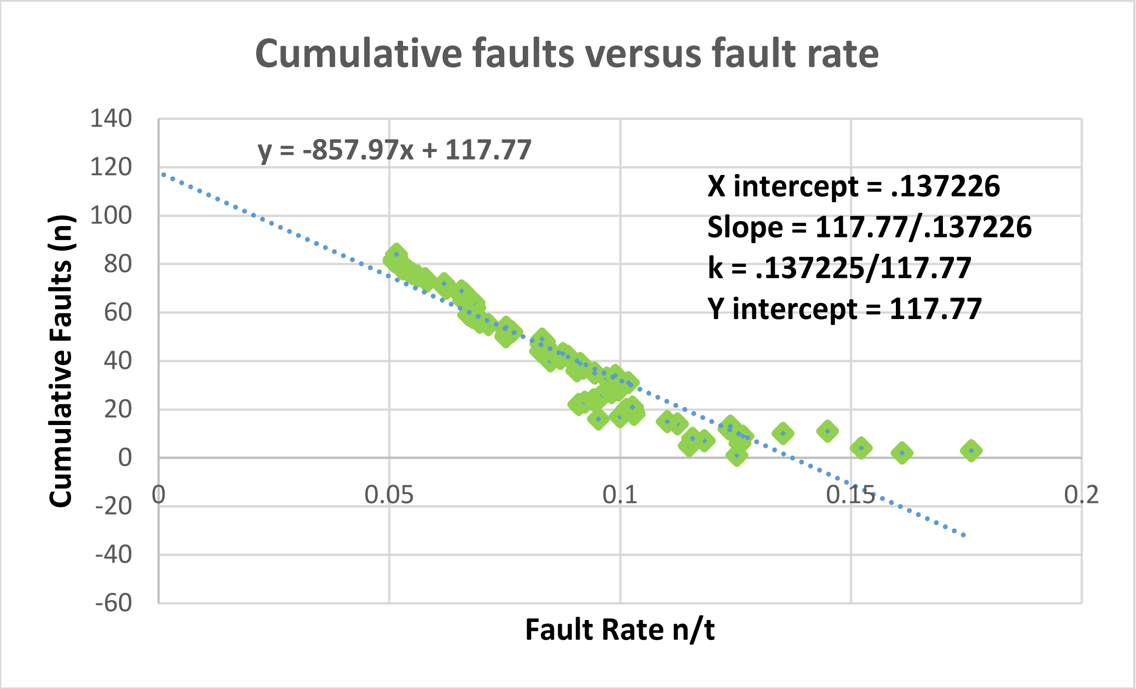
Some of the simpler models

Model	Estimated remaining defects	Estimated current failure rate	Estimated current MTBF	Estimated current reliability
Musa Basic	N_o - n	$l(n) = l_o (1-(n/N_{o)})$	The inverse of the estimated failure rate	$e^{-(l(n) * \text{mission time})}$ Or $e^{-(l(t) * \text{mission time})}$
Jelinski-Moranda		$\lambda(n)= k(N_o-n)$		
Goel-Okumoto		$l(t) = N_o k e^{-kt}$		



n – cumulative defects discovered in testing to date

t – cumulative hours of operation in testing to date



EXAMPLE WITH REAL DATA

n=84 defects discovered to date

T=1628 operational test hours to date

Example

Model	**Estimated remaining defects**	**Estimated current failure rate in terms of failures per hours**	**Estimated current MTBF in hours**	**Estimated current reliability as a function of 8 hours of mission time**
Musa Basic	$\hat{N}_0 - n =$ 117.77-84 = 34 So 71% of the defects estimated have been removed.	$\hat{\lambda}(n) = \hat{\lambda}_0(1-(n/\hat{N}_0) =$ .137226*(1-84/117.77) = .03935	25.41366 hours	$e^{-(.03935 * 8)}$ = .772993
Jelinski-Moranda		$\hat{\lambda}(n) = \hat{k}(\hat{N}_0\text{-}n) =$ .001165*(117.77-84) = .03934	25.4181 hours	$e^{-(.03934 * 8)}$ = .772999
Goel-Okumoto		$\hat{\lambda}(t) = \hat{N}_0 \hat{k} e^{-\hat{k}t} =$ 117.77*.001165*e(-.001165*1628) = .02059	48.56585 hours	$e^{-(.02059 * 8)}$ = .84813

Notice that 2 of the models have the same result. That's because the models use different unknowns which are based on the same assumptions. Only one of them needs to be used by the practitioner.

Forecasting test hours needed to reach a specific objective

Δt = additional test duration = $(N_0/\lambda_0)^* \ln(\lambda_0/\lambda_f)$

Where:

- Δt is the number of test hours required to meet the objective
- N_0 is the estimated inherent defects
- λ_0 is the initial failure rate (the actual very first observed failure rate from the first day of testing)
- λ_p is the objective or desired failure rate

Once the Δt is computed, it should be divided by the number of work hours per day or week to determine how many more days or weeks of testing are required to meet the objective.

Conclusions

Software reliability can be predicted before the code is written using prediction/assessment models

- 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

Software reliability can be estimated during testing using the reliability growth models

- Used to determine when to stop testing
- Used to quantify effort required to reach an objective
- Used to quantify staffing required to support the software once deployed

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 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, 2019
- [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 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
  - Failure rate is from system or end user's perspective
- KSLOC
  - 1000 source lines of code – common measure of software size

MISSION READY SOFTWARE

SOFTREL LLC

HTTP://WWW.MISSIONREADYSOFTWARE.COM

SALES@SOFTREL.COM

321-514-4659