Software Reliability Engineering

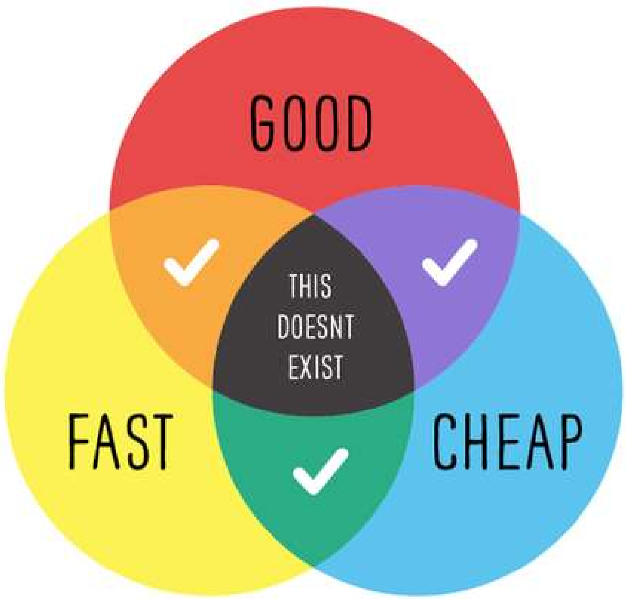
Dr. Tahir Jameel

Outline

- Self Introduction
- Impact of Software Defects
- Software Reliability
- Software Reliability Models
  - Prediction Models
  - Estimation Models
  - Reliability Analysis on a Case Study
- Conclusion

Impact of Software Defects

Dimensions of a Software Project



Fast, Good, Cheap: Pick any 2

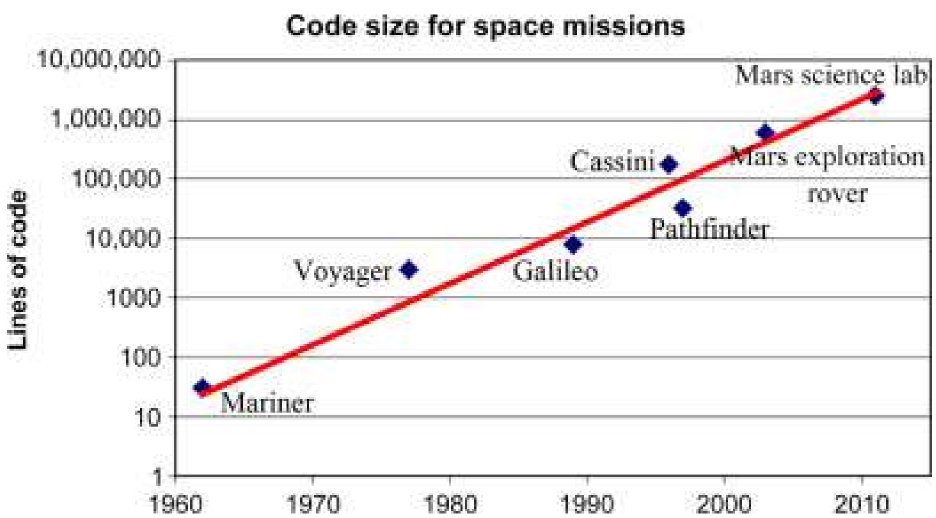
That means there's always a trade off:

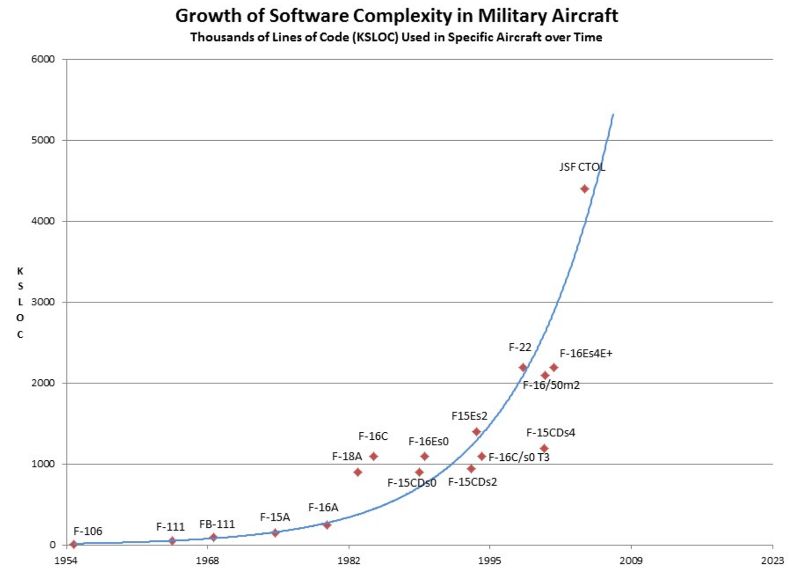
- **Cheap + fast** = lower quality work
- **Fast + good** = expensive
- **Good + cheap** = not happening anytime soon

What are your priorities?

Courtesy: https://medium.com/@devsociety_/good-cheap-fast-pick-two-and-how-ngos-can-play-the-triangle-like-a-pro-20d1380884a8

Code Size and Complexity is Increasing





F-35 have about 24 Millions Line of Code

Ref: Abella, J., & Cazorla, F. M. (2017). Harsh computing in the space domain. In *Elsevier eBooks* (pp. 267–293). https://doi.org/10.1016/b978-0-12-802459-1.00009-9

Ref: Exponential Growth of System Complexity, System Architecture Virtual Integration. Available at: https://savi.avsi.aero/about-savi/savi-motivation/exponential-system complexity/ (Accessed: 22 June 2023).

Increased Defects Fixation Cost

- Industrial Average: about 15 – 50 errors per 1000 lines
- Microsoft Applications: about 10 – 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per KLOC in production

> " One report cites a leaked Microsoft memo stating that Windows 2000 has 63,000 known bugs. Microsoft says the bugs, most of which are trivial, are not a problem" CNN 2000

- It is possible to achieve zero defects but it is also costly.
- NASA was able to achieve zero defects for the Space Shuttle Software, but at a cost of thousands of dollars per line of code.

1. *Ref: McConnell, Steve. Code complete. Pearson Education, 2004.*
2. *Ref: Exponential Growth of System Complexity* (no date) *System Architecture Virtual Integration*. Available at: https://savi.avsi.aero/about-savi/savi-motivation/exponential-system-complexity/ (Accessed: 22 June 2023).
3. *Ref: Will bugs scare off users of new Windows 2000? (2000) CNN. Available at: http://edition.cnn.com/2000/TECH/computing/02/17/windows.2000/ (Accessed: 18 June 2023).*

Financial Losses due to Faulty Software

- According to a report by NIST (USA), faulty software costs **$59.5 Billion** annually to US economy (**2002**)
- A research at Cambridge University (**2013**) showed that the global cost of software bugs is **$312 billion USD** annually (Approx.)
- According to the Consortium for Information and Software Quality, poor software quality cost US companies **$2.08 trillion** (**2020**)

Goebelbecker, E. (2022, May 9). *How much could software errors be costing your company?*. RAYGUN. https://raygun.com/blog/cost-of-software errors/#:~:text=According%20to%20the%20Consortium%20for,software%20errors%20in%20legacy%20systems.

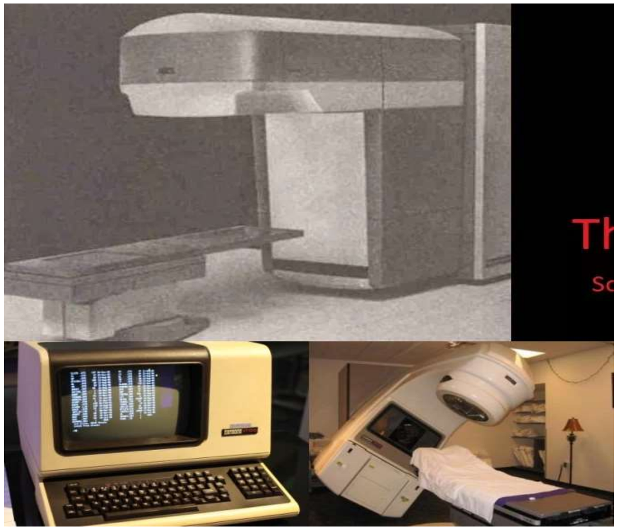
The Explosion of Ariane 5 1996

- Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off
- A 64-bit floating point number was converted to a 16-bit signed integer
- The number was larger than 32,767, the largest integer storable in a 16 bit signed integer, and thus the conversion failed
- Result loss of development cost $7 B. Rocket and its cargo were valued at $500 M

Ref: Lions, Jacques-Louis. "Flight 501 failure." *Report by the Inquiry Board* 190 (1996)

Therac-25 1986

- A radiation therapy machine
- It was involved in at least six accidents
- Concurrent programming error
- It sometimes gave its patients radiation doses that were hundreds of times greater than *normal
- Result: Serious injury and even loss of life



Ref: Leveson, Nancy G., and Clark S. Turner. "An investigation of the Therac-25 accidents." *Computer* 26.7 (1993): 18-41

Investigation of Accidents

- The Therac-20, a predecessor of the Therac-25, employed independent protective circuits and mechanical interlocks to protect against overdose.
- The Therac-25 relied more heavily on software.
- Moreover, when the manufacturer started receiving accident reports, it was unable to reproduce the accidents, assumed hardware faults, implemented minor fixes, and then declared that the machine's safety had improved by several orders of magnitude.

Ref: Leveson, Nancy G., and Clark S. Turner. "An investigation of the Therac-25 accidents." *Computer* 26.7 (1993): 18-41

Coding Errors only?

- Requirement Engineering (~= 20%)
- Software Design (~= 30%)
- Code/ Others (~= 50%)

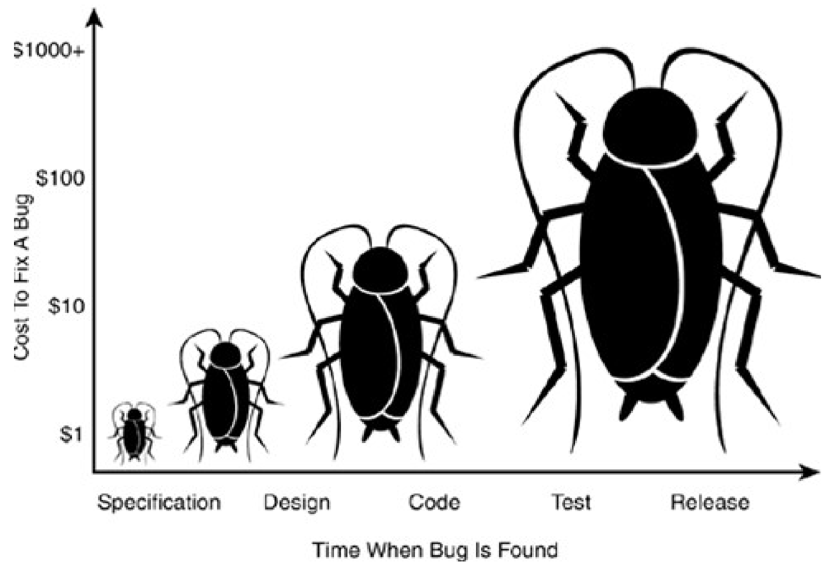
Book Ref: Integrated Approach to Software Engineering P. Jalote 2nd Edition

Causes of Common Bugs

- Lack of available calendar time/resources to find all of the defects that can result in failures
- Exceedingly complex event driven systems that are difficult to conceptualize and therefore implement and test
- Organizational culture that neglects to support sufficient rigor, skills, or methods required to find the defects
- Technical decisions that result in incorrect architecture or design decision that cannot support the stakeholders specifications
- Insufficient project or risk management that leads to schedule delays that lead to less time for reliability testing
- Operations—Contract issues, interoperability due to bad specifications and stakeholder communications

Ref: Neufelder, A. (2016). IEEE Recommended Practice on Software Reliability. *IEEE Standard*, 1633-2016.

Relative Cost of Bugs



Bugs fixed earlier cost less

> **50% of my company employees are testers and the rest spend 50% of their time testing!**

Bill gates 1995

Can we Quantify Software Quality?

- “Capability of a software product to conform to requirements.“ ISO
- Functional quality is typically assessed dynamically but it is also possible to use static tests
  - When to stop testing?
- "You cannot control what you cannot measure.“ (Tom DeMarco)
  - But how to quantify quality of software? Quality in numbers?
- How much a user can depend on a Software?
  - What can be the actions to enhance dependability of software?
- The answers of these questions is “Software Reliability”

Software Reliability

Software Reliability

"The ability of a system or component to perform its required functions under stated conditions for a specified period of time."

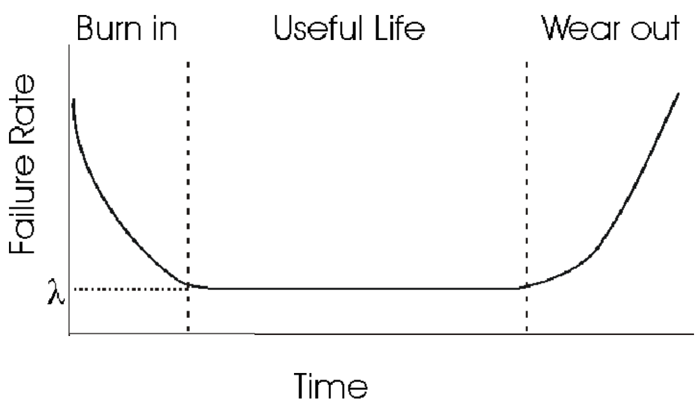
IEEE Std 610.12-1990

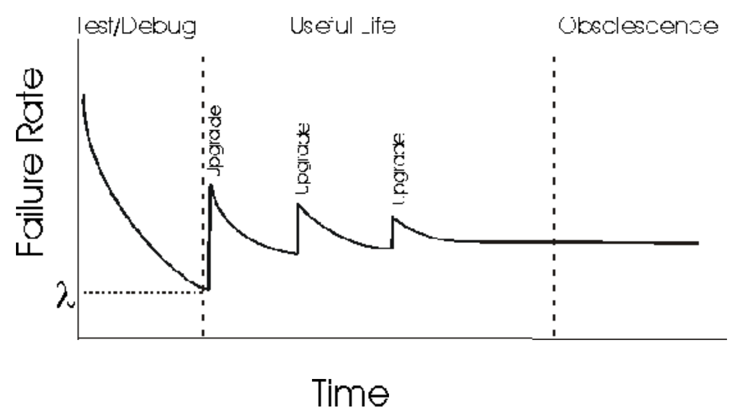
- Examples:
  - An aircraft should fly for six hours (mission time) without any failure after fueling.
  - The system must perform without failure in 95 percent of use cases during a month

System Reliability

Hardware Reliability × Operator Reliability × Software Reliability = System Reliability

- Hardware Reliability → Manufacturing Perfection
- Operator Reliability → Operator Training/ SOP
- Software Reliability → Design Perfection





Uniqueness of Software Reliability

- Non-tangible, complex operations
- Not manufactured but designed, no faults such as machining, processing etc.
- Software does not wear out as a function of calendar time but usually obsolete
- Fault discovery is related to how much the software is exercised
- Software has frequent changes → requirements change, bug fix, improvement
- Fixing of software bug may introduce a potential defect
- Software Reliability increases with time whereas in Hardware; reliability decreases due to aging
- Evaluation of Software reliability is entirely different from hardware reliability

Software Reliability Models

IEEE 1663:2016

Reliability Models

- Software quality is the most challenging aspect of software development industry and Reliability is one of the key aspects of software quality.
- Reliability is evaluated at different stages of development life cycle and the models are divided into two types:
  - **Prediction**: Used in early stages of SDLC when software is not developed/ testable, is based on metrics and historical data.
  - **Estimation/Growth**: Used when software is developed, is based failure data collected during testing

Prediction Models

IEEE 1663:2016

Prediction Models

- Predictive models can predict software reliability in early stages of SDLC based on software metrics and size
- Initially, reliability growth estimation models were proposed
- Reliability estimation (use failure data) is very late in SDLC
  - to enhance software reliability we have to redesign the software
  - cost of redesign/ rework is very high
- Prediction model was proposed by Rome Lab in 1978 to predict reliability early phases of SDLC
  - before a testable software is developed
  - actions can be taken preemptively to achieve desired reliability

Prediction Models…

- Use actual historical data from real software projects.
- The user answers a list of questions which calibrate the historical data to yield a software reliability prediction.
- The accuracy of the prediction depends on:
  - How many parameters (questions) and datasets are in the model.
  - How current the data is.
  - How confident the user is of their inputs.

Prediction Models

Model	Inputs	Predicted Output	Industry Supported	Year Developed/ Last Updated
Industry Tables	1	Defect Density	Several	1992, 2015
CMMI Tables	1	Defect Density	Any	1997, 2012
Shortcut model Neufelder	23	Defect Density	Any	1993, 2012
Full-scale model Neufelder	94-299	Defect Density	Any	1993, 2012
Historical data	Minimum 2	Defect Density	Any	NA
RADC TR-92- 52	43-222	Defect Density	Aircraft	1978, 1992

Ref: Neufelder, A. (2016). IEEE Recommended Practice on Software Reliability. *IEEE Standard*, 1633-2016.

Steps to Predict Software Reliability

- Step 1. Predict the defect density
  - The size is predicted so as to yield the total predicted defects
- Step 2. The fault profile is predicted.
  - How software will be deployed and its duty cycle
- Step 3. The failure and MTBF, MTBCF are predicted.
- Step 4. Predict reliability
- Step 5. Predict availability

Predict the Defect Density

- We used Neufelder's Shortcut Model in our surveys
- It predicts defect density of the software under test (no. of defects/ KLOC)
- Shortcut model is based on 23 Questions related to Strengths and Risks of the software under test
- Based on the survey, the model predicts defect density (DD)
  - Strengths-Risks ≥ 4 predicted DD = 0.110
  - Strengths-Risks ≤ 0.5 predicted DD = 0.647
  - Otherwise DD = 0.239
- Total predicted defects = DD x Total Size (predicted)

Ref: Neufelder, A. (2016). IEEE Recommended Practice on Software Reliability. *IEEE Standard*, 1633-2016.

Table 47 —Shortcut model survey

Strengths	
1	We protect older code that should not be modified.
2	The total schedule time in years is less than one.
3	The number of software people years for this release is less than seven.
4	Domain knowledge required to develop this software application can be acquired via public domain in short period of time.
5	This software application has imminent legal risks.
6	Operators have been or will be trained on the software.
7	The software team members who are working on the same software system are geographically co-located.
8	Turnover rate of software engineers on this project is < 20% during course of project.
9	This will be a maintenance release (no major feature addition).
10	The software has been recently reconstructed (i.e., to update legacy design or code).
11	We have a small organization (<8 people) or there are team sizes that do not exceed 8 people per team.
12	We have a culture in which all software engineers value testing their own code (as opposed to waiting for someone else to test it).
13	We manage subcontractors: outsource code that is not in our expertise, keep code that is our expertise in house.
14	There have been at least four fielded releases prior to this one.
15	The difference between the most and least educated end user is not more than one degree type (i.e., bachelors/masters, high school/associates, etc.).
Risks	
1	This is brand new release (version 1), or development language, or OS, or technology (add one for each risk).
2	Target hardware/system is accessible within 0.75 points for minutes, 0.5 points for hours, 0.25 points for days, and 0 points for weeks or months.
3	Short term contractors (< 1 year) are used for developing line of business code.
4	Code is not reused when it should be.
5	We wait until all code is completed before starting the next level of testing.
6	Target hardware is brand new or evolving (will not be finished until software is finished).
7	Age of oldest part of code >10 years.

Ref: Neufelder, A. (2016). IEEE Recommended Practice on Software Reliability. *IEEE Standard*, 1633-2016.

Data for Reliability Prediction Analysis

- Data for different case studies from different software houses in Islamabad was collected for Reliability Prediction analysis
- This data was then used to predict the reliability of software systems using Shortcut Model.
  - Case Study 1 (URR)
  - Case Study 2 (LM)
  - Case Study 3 (WW)
  - …

Case Study 1

Project Name	Version	Version Date
URR	Confidential	April, 2023
Size of Code	7000 KSLOC	
Defect Density (Shortcut Model)	0.11	
Total Number of Defects	770	
Defects in 1st Month	90.4	
Failure Rate / Hour	0.6	
MTBF (1st Month)	1.65	
Availability(1st Month)	40.8%	
Reliability(1st Month)	88.6%	
Reliability(2nd Month)	89.9%	

link

Case Study 2

Project Name	Version	Version Date
LM	41	17 May, 2023

Size of Code	220 KSLOC + 2MB DLLs
Defect Density (Shortcut Model)	0.11
Total Number of Defects	91.52
Defects in 1st Month	10.75
Failure Rate / Hour	0.105
MTBF (1st Month)	9.48
Availability(1st Month)	93.3%
Reliability(1st Month)	99.5%
Reliability(2nd Month)	99.6%

link

Case Study 3

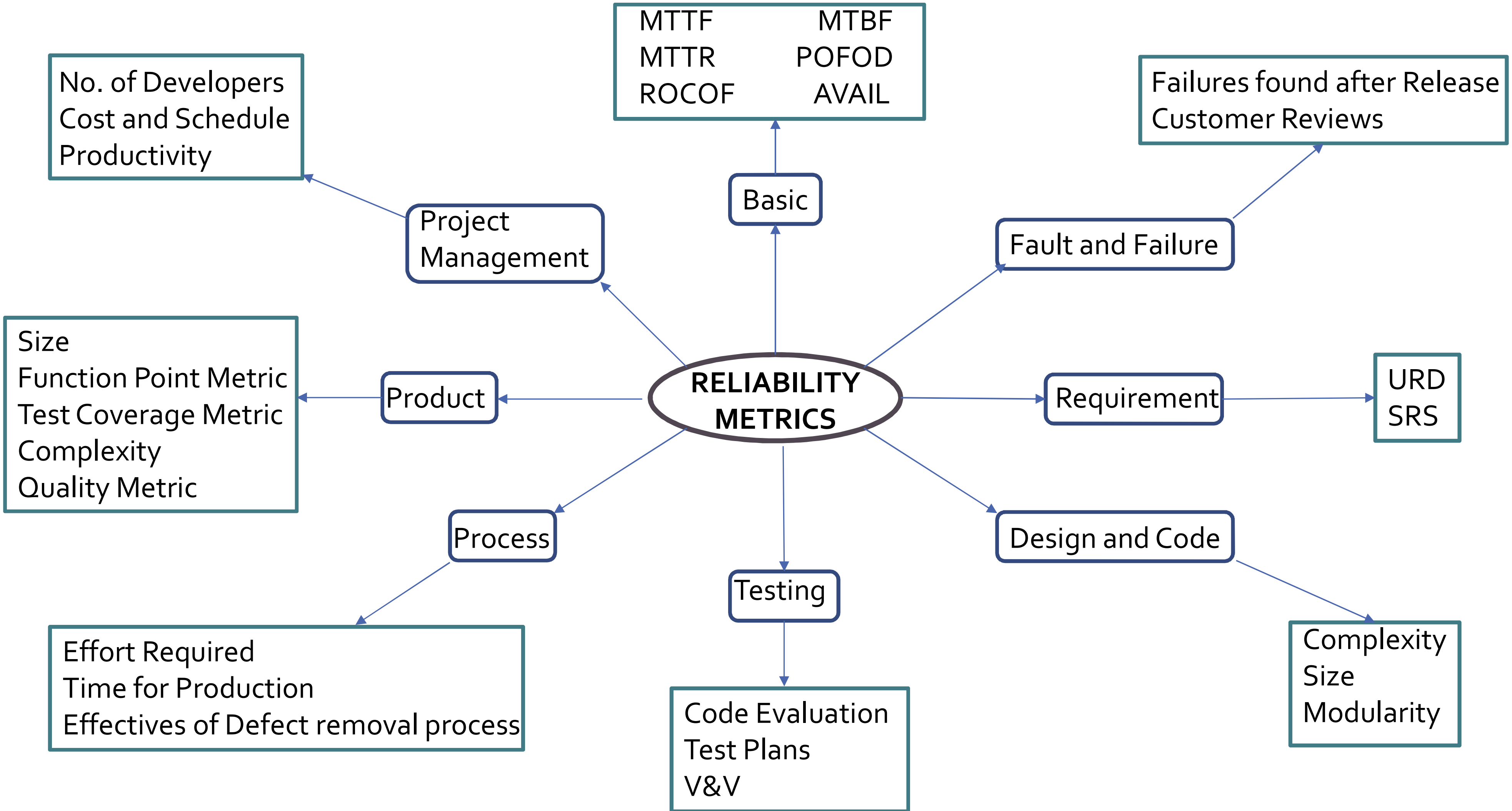
Project Name	Version	Version Date
WW	1.9	5 April, 2023

Size of Code	299 KSLOC + 301MB DLLs
Defect Density (Shortcut Model)	0.239
Total Number of Defects	22084
Defects in 1st Month	5414
Failure Rate / Hour	30.079
MTBF (1st Month)	0.033
Availability(1st Month)	25.2%
Reliability(1st Month)	86.2%
Reliability(2nd Month)	89.4%

link

Research Directions

- AI based software reliability prediction model
  - Challenge is availability of data
- Augmenting Rome Lab Model
  - Unifying related metrics
  - Adding new metrics

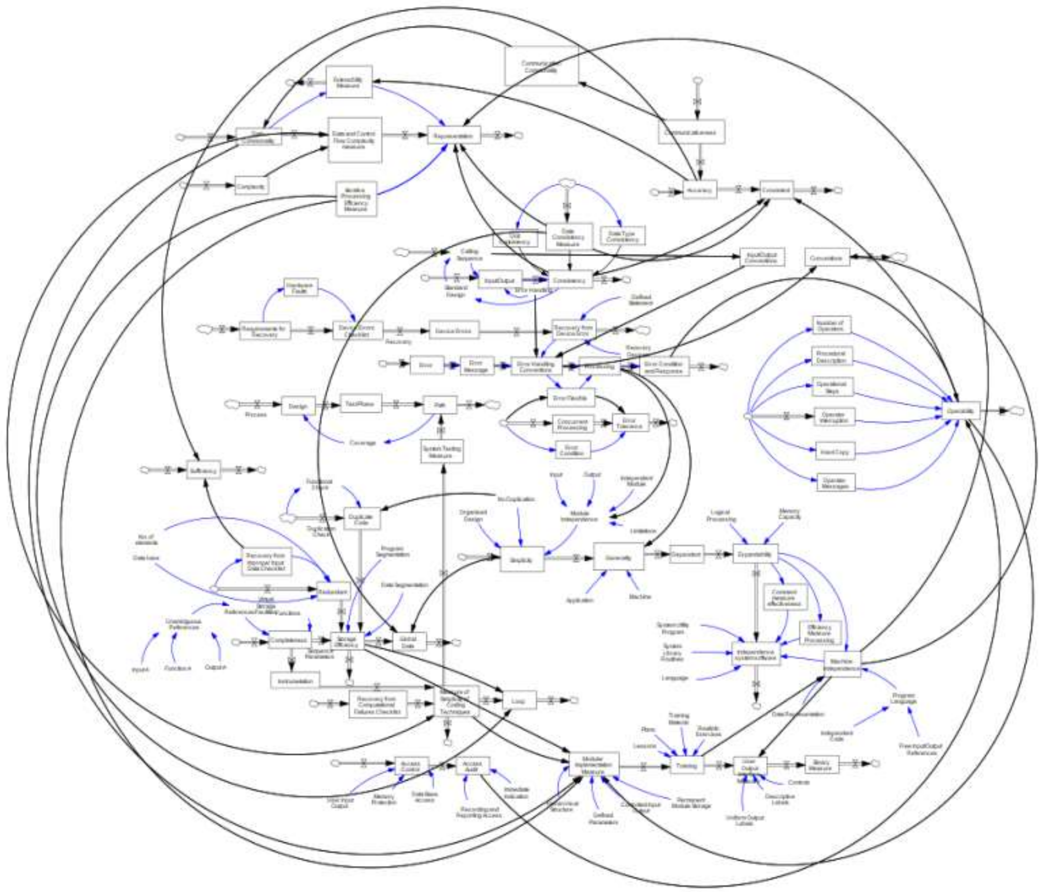


RELIABILITY METRICS
Basic
MTTF MTBF
MTTR POFOD
ROCOF AVAIL
Project Management
No. of Developers
Cost and Schedule
Productivity
Fault and Failure
Failures found after Release
Customer Reviews
Product
Size
Function Point Metric
Test Coverage Metric
Complexity
Quality Metric
Requirement
URD
SRS
Process
Effort Required
Time for Production
Effectives of Defect removal process
Testing
Code Evaluation
Test Plans
V&V
Design and Code
Complexity
Size
Modularity

Rome Lab Model

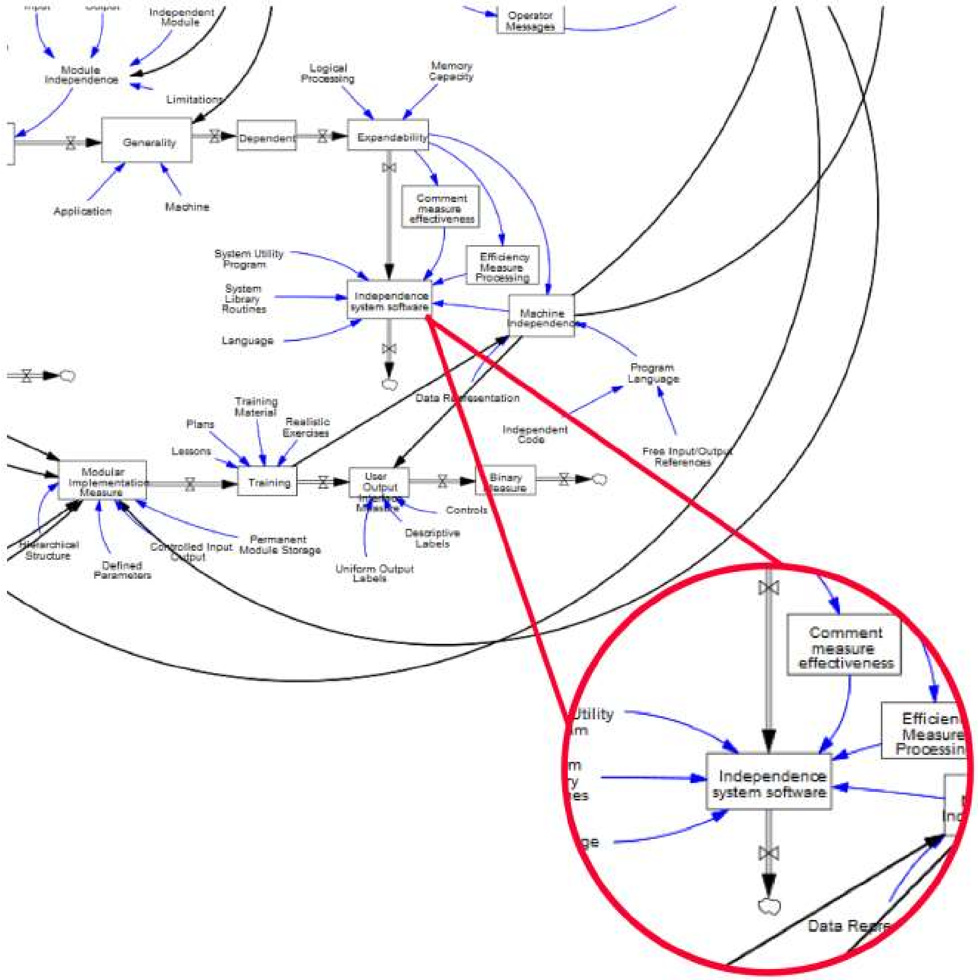
- It is composed of 172 different questions
- Factors of Rome Lab are
  - Application type
  - Development
  - Complexity
  - Traceability
  - Anomaly
  - Quality review and
  - Standard review

Determine the number of modules with complexity >= 20
Determine the number modules complexity >= 7 and <20
Determine the number modules complexity < 7



System dynamic model (Vensim) to determine interconnection between the metrics

Rome Lab Model metrics and their variables interlinked with each other with direct and indirect relationship


Independent Module
Module Independence
Limitations
Operator Messages
Logical Processing
Memory Capacity
Generality
Dependent
Expandability
Application
Machine
Comment measure effectiveness
System Utility Program
Efficiency Measure Processing
System Library Routines
Independence system software
Machine Independence
Language
Program Language
Data Representation
Training Material
Plans
Realistic Exercises
Lessons
Independent Code
Free Input/Output References
Modular Implementation Measure
Training
User Output Interface Measure
Binary Measure
Controls
Descriptive Labels
Hierarchical Structure
Defined Parameters
Controlled Input Output
Permanent Module Storage
Uniform Output Labels
Comment measure effectiveness
Independence system software
Data Repre

Advantages of Reliability Prediction

- Reliability Prediction gives us an idea about the reliability that can be achieved based on current information
- The actions can be taken proactively to improve the reliability of the software product to achieve the desired reliability
- Predicted Defect Density gives a quality measure of development and testing process
- Number of defects predicted is a theoretical target for testing team, it can help to decide (along with other factors), “when to stop testing”?
- Factors contributing the defect density can be identified and to lower the defect density sensitive areas can be augmented which directly contributes the reliability

Growth Estimation Models

IEEE 1663:2016

Growth Models

- Growth models are based on failure data during testing phase to forecast the failures in future. They are used with an assumption that the reliability of the system improves after testing.
- They provide information of:
  1. Time for the occurrence of next faults
  2. Number of faults expected in software life
  3. Reliability
  4. Additional testing required to achieve a certain level of reliability
  5. Release of Software
  6. When to stop testing
- Examples Goel-Okumoto, Basic Execution Time Model etc.

Need of Software Reliability Models

- It is therefore necessary to develop a practical and applicable model that can determine:
  - Software failure growth trend
  - Predict the number of failures and software reliability given a specific period of operating time
  - Propose the optimal release time of new products
  - Schedule the delivery time for the next release based on the reliability level of previous release.

Prediction vs Growth Model

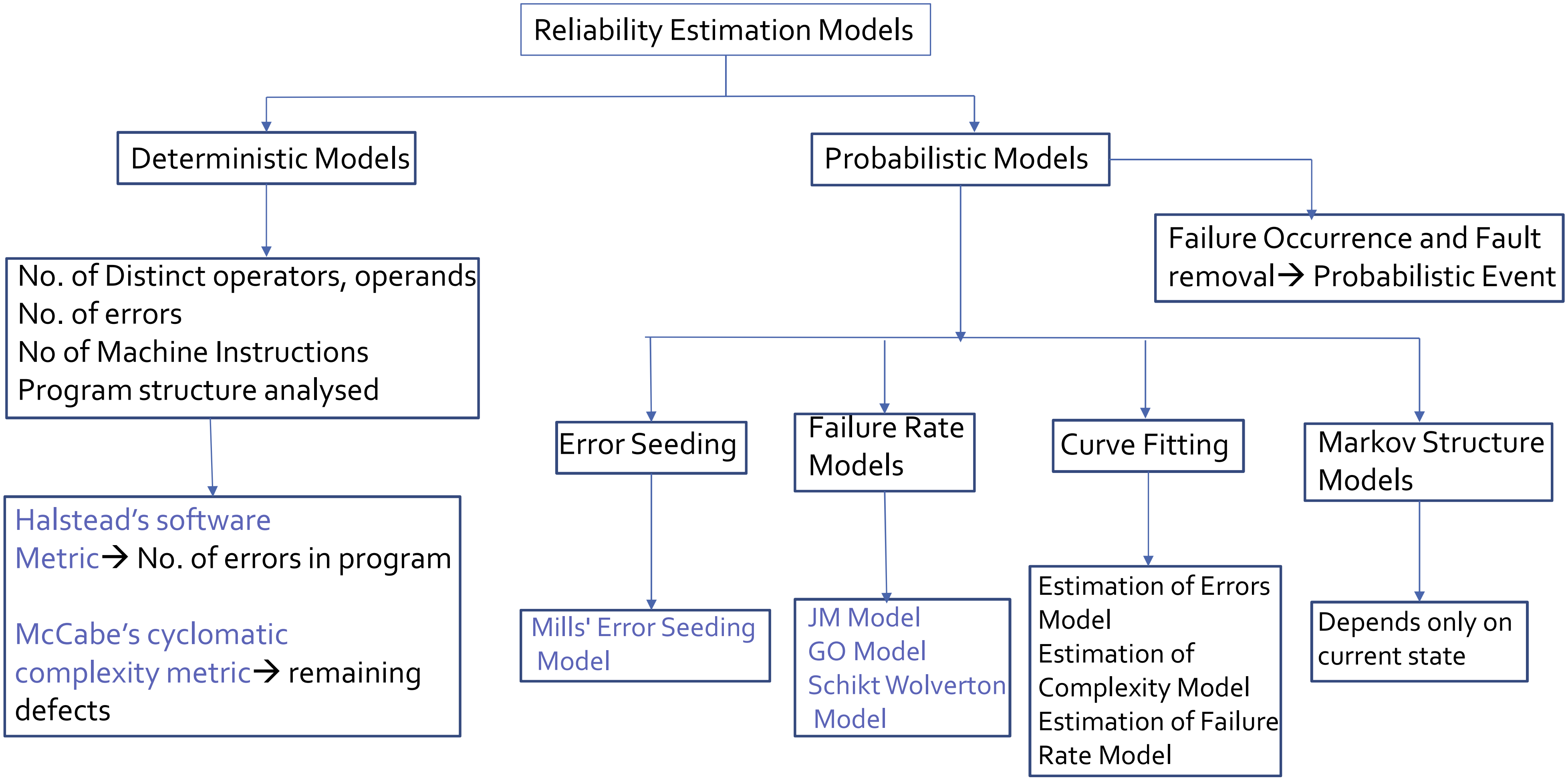
Comparison attribute	Software reliability prediction models	Software reliability growth models
Used during this phase of development	Any phase as long as the scope of the software is defined	After software system testing commences
Inputs	— Indicators such as development practices, personnel, process, inherent risks — Size — Duty cycle — Expected reliability growth	— Defects observed per time or defects observed during some internal. — Testing hours per interval.
Outputs	Predicted defects, failure rate, reliability, availability	
Benefits	— Allows for a sensitivity analysis before the software is developed — Allows for determination of an allocation before the software is developed — Identifies risks early — Useful for assessing vendors	— Identifies when to stop testing — Identifies how many people are needed to reach an objective — Validates the prediction models
Model framework	Uses empirical data from historical projects in which the development practices and the delivered defects are known	Uses various statistical models to forecast based on the current trend

Ref: Neufelder, A. (2016). IEEE Recommended Practice on Software Reliability. *IEEE Standard*, 1633-2016.

Estimation Models

Software Reliability Estimation Models

- Software failure data collected during software testing phase is used to compute reliability of the system.
- Different software reliability models are applied on the data to give statistical estimations of the reliability
- Software reliability Estimation models are classified into two types:
  - Deterministic Models
  - Probabilistic Models


Reliability Estimation Models
Deterministic Models
Probabilistic Models
Failure Occurrence and Fault removal→ Probabilistic Event
No. of Distinct operators, operands
No. of errors
No of Machine Instructions
Program structure analysed
Halstead's software Metric→ No. of errors in program
McCabe's cyclomatic complexity metric→ remaining defects
Error Seeding
Failure Rate Models
Curve Fitting
Markov Structure Models
Mills' Error Seeding Model
JM Model
GO Model
Schikt Wolverton Model
Estimation of Errors Model
Estimation of Complexity Model
Estimation of Failure Rate Model
Depends only on current state

Deterministic Models

- Deterministic models study:
  1. Distinct operators of a program, operand errors and instructions.
  2. The branches in a program to study its control flow.
  3. The data flow of program (Data sharing and passing)
- Well known deterministic models are Halstead's software metric and McCabe's Cyclomatic complexity metric
- Provided innovative quantitative approach to measure quality.
- Doesn't consider random events, so not suitable for modern software.

Probabilistic Models

- These models consider failure detection and failure removal as probabilistic models.
- Classification of these models is:
  - Error Seeding
  - Failure Rate
  - Curve Fitting
  - Reliability Growth
  - Markov Structure
  - Time Series
  - NHPP.

Software Reliability Growth Models (SRGM)

- SRGM helps in rendering a balanced amalgamation in terms of expense, reliability, productivity and performance.
- A SRGM follows NHPP distribution that gives an estimated count of faults for both calendar as well as running timeline.

Basic Assumptions

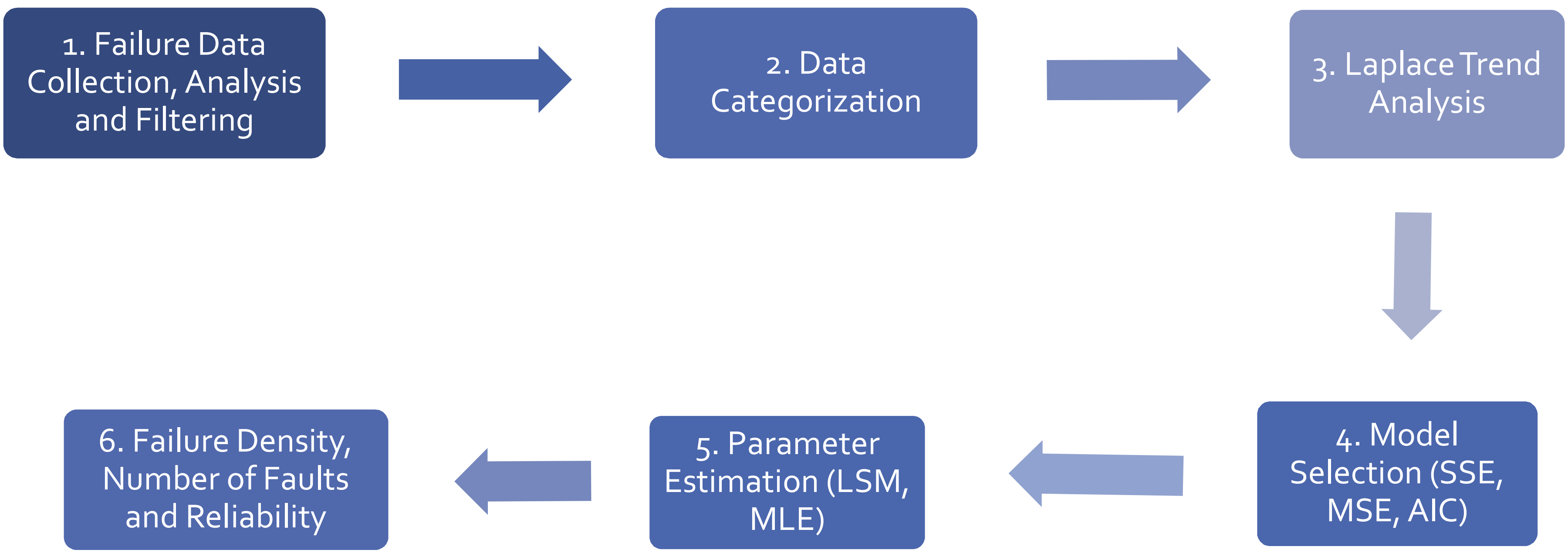
- The program contains N initial faults which is an unknown
- Each fault in the program is independent and equally likely to cause a failure during a test.
- Time intervals between occurrences of failure are independent of each other.
- Whenever a failure occurs, a corresponding fault is removed with certainty.
- The fault that causes a failure is assumed to be instantaneously removed, and no new faults are inserted during the removal of the detected fault.

Model	Program Failure Rate	Reliability	Parameters	Assumptions
Jelinski Moranda	$\lambda(t_i) = \phi[N-(i-1)]$,	$R(t_i) = e^{-\phi(N-i+1)t_i}$	**ϕ** = a proportional constant, the contribution any one fault makes to the overall program ***N*** = the number of initial faults in the program ***ti*** = the time between the (i-1)*th* and the i*th* failures	Software Failure rate is initially a constant and is proportional to remaining faults in the program
Geometric	$\lambda(t_i) = Dk^{i-1}$	$R(t_i) = e^{-Dk^{i-1}ti}$	**D**= Initial program failure rate **k**= parameter of geometric function	Software Failure rate is initially a constant and decreases geometrically with time
Goel Okumoto	$\lambda(t_i)=abe^{-bt}$	$\hat{R}(x\|t) = e^{-\hat{a}[e^{-\hat{b}t}-e^{-\hat{b}(t+x)}]}$	**a**= expected total number of faults before testing **b**= failure detection rate/ failure Intensity	Software Failure detection rate is time dependent and constant.
Delayed S shaped	$\lambda(t) = ab^2te^{-bt}$	$R(s\|t) = e^{-a[(1+bt)e^{-bt}-e^{-(1+b(s+t))}]e^{-b(t+s)}}$	**a**= expected total number of faults before testing **b**= failure detection rate/ failure Intensity	Software Failure rate is time dependent and differs among faults.
Weibull	$f(t) = \frac{\beta(t-\gamma)^{\beta-1}}{\theta^\beta} e^{-\left(\frac{t-\gamma}{\theta}\right)^\beta}$	$R(t) = e^{-\left(\frac{t-\gamma}{\theta}\right)^\beta}$	**Ɵ**= Scale Parameter **β** = Shape Parameter **γ** = Location Parameter	Used for both hardware and software systems Fluctuating Hazard Rate function

Application of SRGM for Reliability Estimation

Steps for Reliability Estimation

- Following are the steps that need to be taken to compute Reliability using SRGM:



SFRAT University of Massachusetts Dartmouth

- SFRAT is Web based software reliability testing suite written in R
- Can take failure data in both interval and time domain format.
- User friendly interface.
- Uses failure data during testing to evaluate reliability and failure intensity.
- Uses 5 well known reliability models to fit on the failure data.
- Can perform all the tasks required for the application of growth models on the testing data.
- Also provides prediction about occurrence of future failures and reliability of system in a known period of time

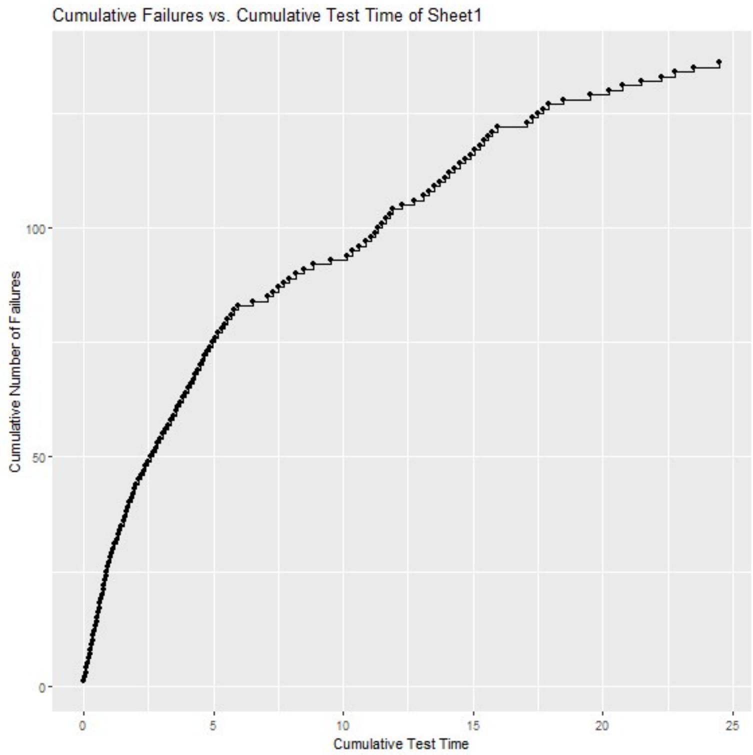
Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/ -

Software Failure Data

- Software Failure Dataset was also acquired for the URR software on which prediction analysis was performed
- The dataset contains total of 136 failures.
- The time for occurrence of each failure is also given in seconds.
- The inter failure time is also computed and added in the dataset
- Small snaps of the used data are given below:

FN	IF	FT
1	3	3
2	30	33
3	113	146
4	81	227
5	115	342
.		
.		
.		
134	1160	82702
135	1864	84566
136	4116	88682

Cumulative Failures vs Time plot



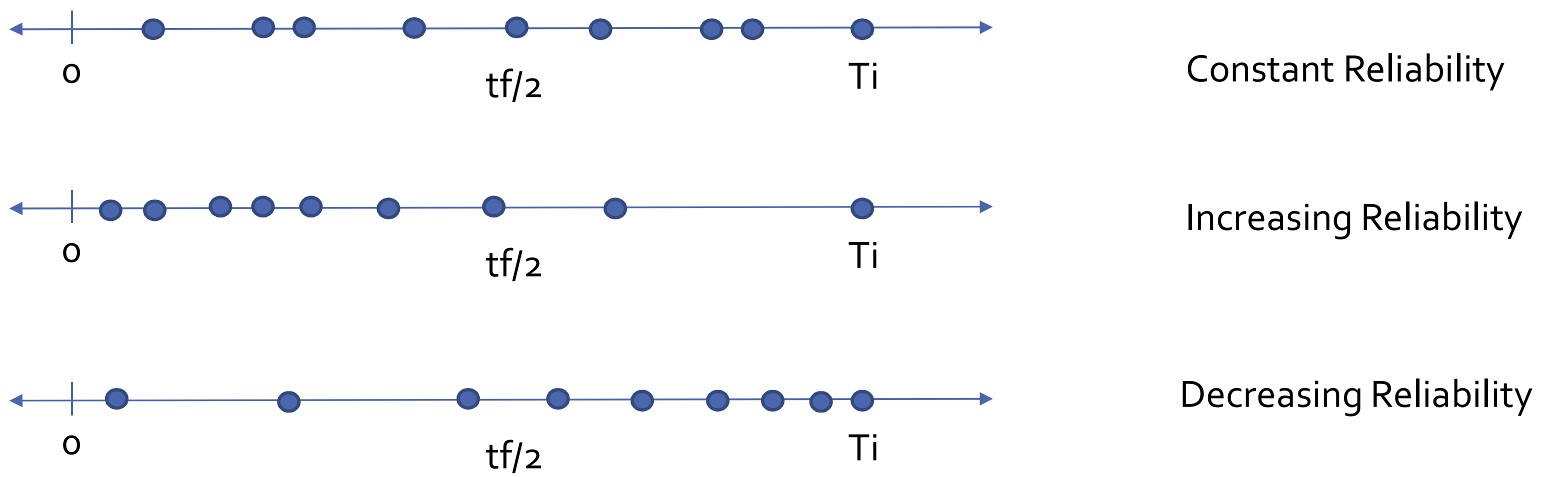
Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

Laplace Trend Analysis

- The Laplace trend test can determine whether the system is deteriorating, improving, or if there is no trend at all.
- When a full-scale reliability program is not in place, the Laplace Test can be used to quantify trends of undesired events for each system element and any combination.
- As a proactive step, this helps management to identify and prioritize elements that need further analysis (e.g., verification, root cause) and possible remedial or corrective action.
- This measure approximates the standardized normal random variable (e.g., z-score).

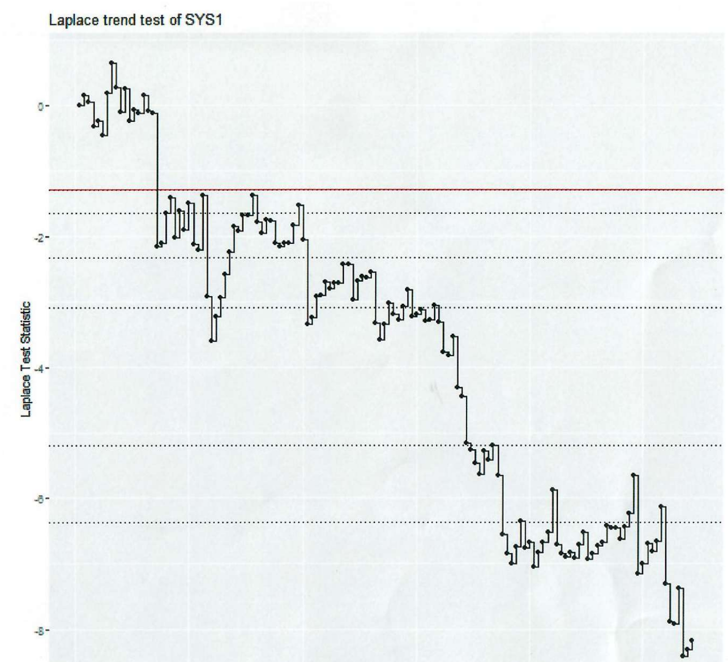
Working

- $UL = \dfrac{\frac{1}{m}\sum_{i=1}^{m} Ti \; - \; \frac{tf}{2}}{tf.\sqrt{\frac{1}{12m}}}$



Decreasing Trend of Software Failures

- Laplace Trend Analysis was done on the dataset and it was found that system shows increasing reliability.



Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

Model Mapping on Test Data

- 5 Reliability Growth models are mapped on the failure data.
- From the mapping and results of AIC, we found out that **Geometric** model is the best fit.

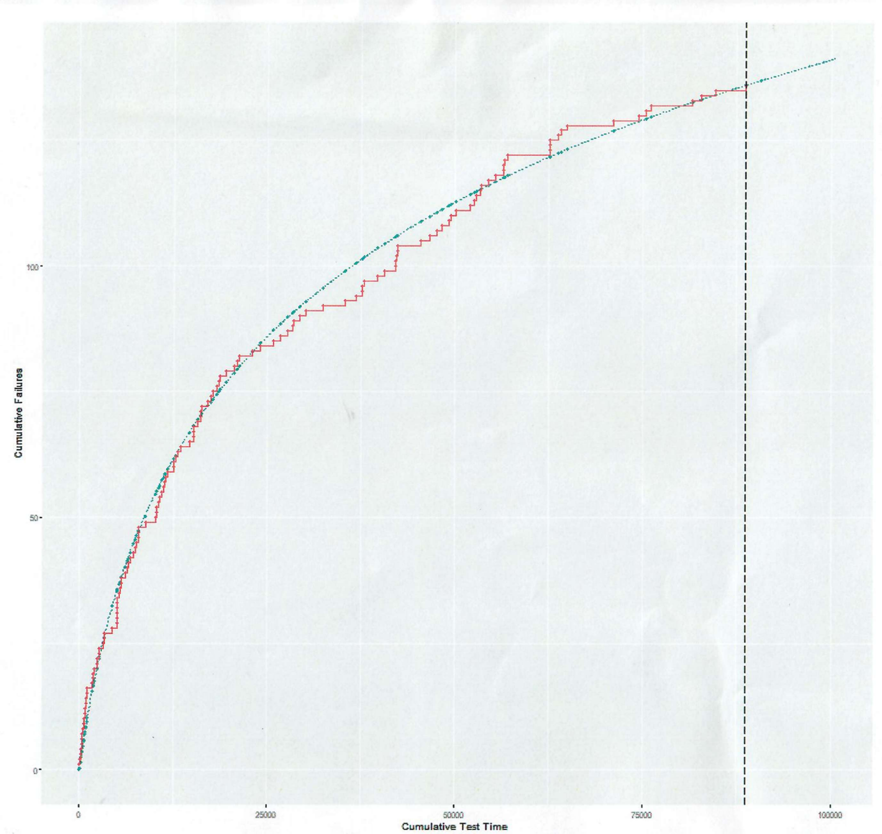
Model	AIC Value
Delayed S Shaped Model	2075
Geometric Model	1937
Goel Okumoto Model	1953
Jelinski Moranda Model	1950
Weibull Model	1938

AIC (Akiake Information Criterion) is used for Model Selection
When a model is used to map given

AIC estimates relative amount of information lost during mapping by all models

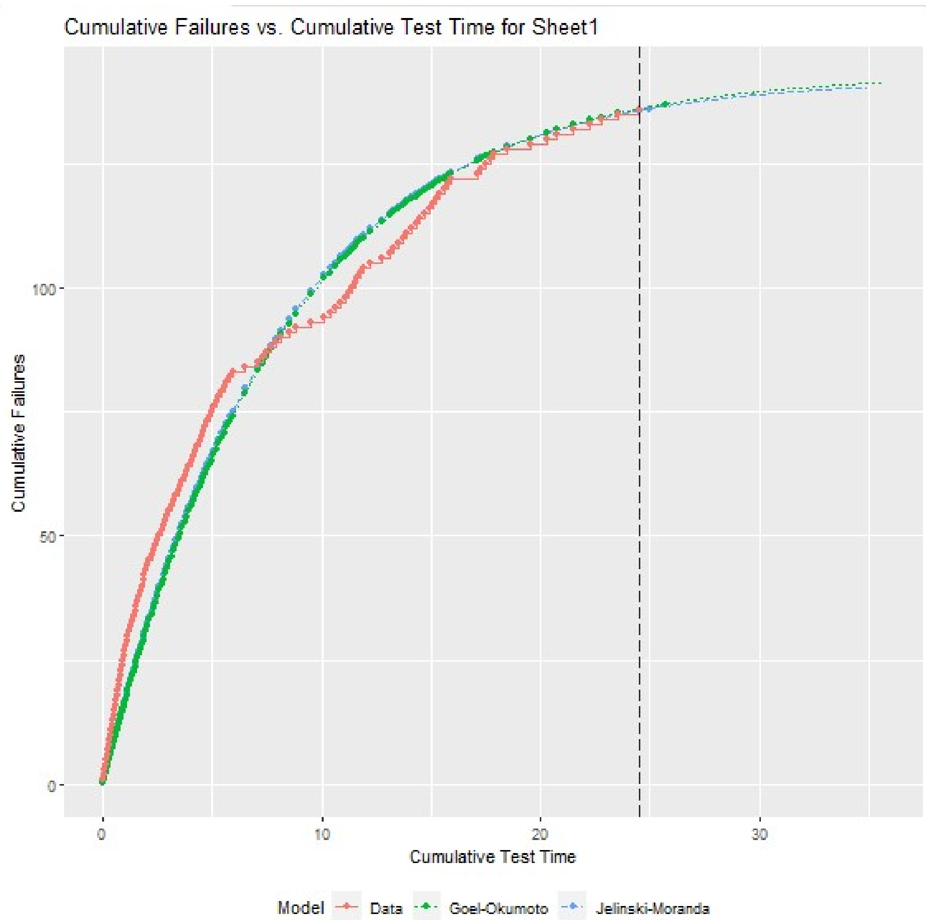
Less information lost, better the model.

Applying Model



Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

Geometric model mapped on the test data (red curve)



Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

Reliability Analysis

- Since the mission time of software is 0.25 hour, so 900 sec time was used for the estimations
- We can get following information from the reliability analysis:
  - The prediction of time of next failures.
  - Reliability of the system at next failures.
  - Defect density of the system at each failure.
  - Time to achieve desired reliability.
  - Expected time of next failure.
  - Number of failures in specified time period.
- In this example, reliability at next 1st failure is predicted.

Reliability Estimation using Geometrics Model

	Failure	GM_D0	GM_Phi	GM_Cum_Time	GM_Cum_Fails	GM_IF_Times	GM_Fail_Intensity	GM_Rel_Growth
	All	All	All	All	All	All	All	All
131	131	NA	NA	75409	129.384177666196	1952.45329102857	0.000544151619872479	0.614478733599268
132	132	NA	NA	76057	129.735356462847	1998.18214547303	0.000539745495836042	0.616892978092519
133	133	NA	NA	81542	132.598815187347	2044.98202586131	0.000505124701708251	0.636203667736553
134	134	NA	NA	82702	133.18082116383	2092.87801693667	0.000498364241985149	0.640046101609262
135	135	NA	NA	84566	134.099923914255	2141.89579095782	0.000487871925807293	0.646056689559348
136	136	0.0106303732336185	0.977114771769902	88682	136.062725552418	2192.06162145936	0.000466198630573915	0.658655903533051
137		NA	NA	90714.4327293947	137	2243.4023973343	0.000456191555114337	0.664557824472022

Showing 131 to 137 of 137 entries

Previous 1 … 10 11 12 13 14 Next

Reliability increases with time

Failure intensity decreases with time

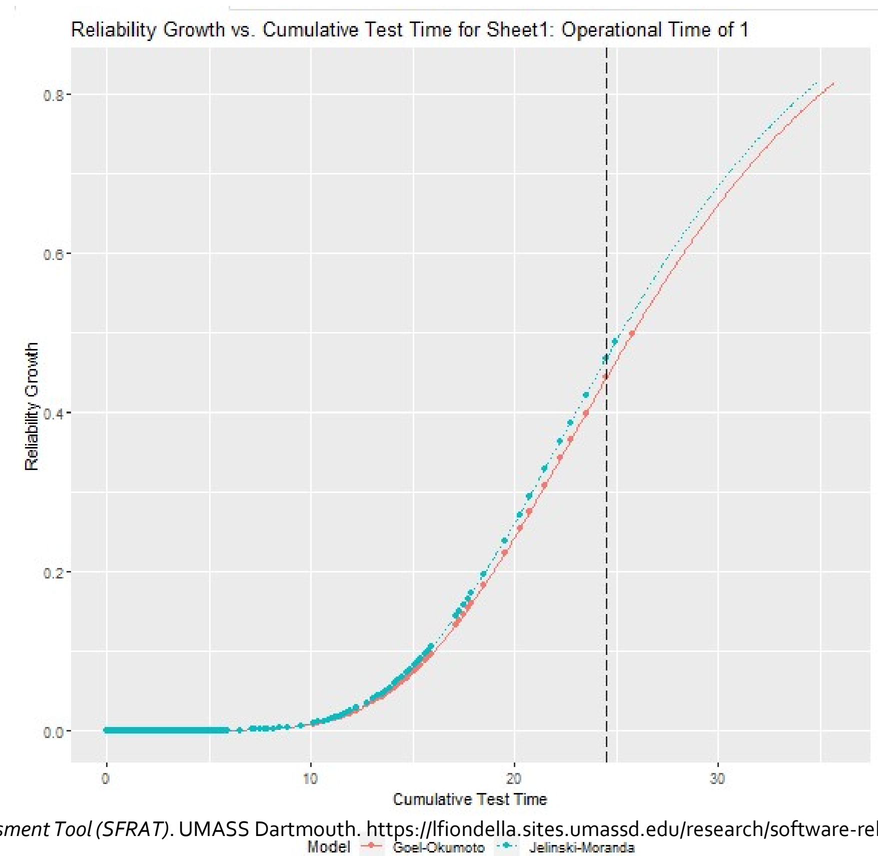
Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

Reliability Analysis of URR (Case Study 1)

- Following are the details of the analysis performed:

Predicted Reliability	**88.6 %**
Estimated Reliability at 136th failure	**65.8%**
Failure Intensity at 136th failure	0.00046
Reliability at 137th failure	66.4%
Failure Intensity at 137th failure	0.00045
Time required to achieve 90% reliability	76 hours
Expected number of failures in next mission time (900 sec)	0.417
Expected time to next Failure	0.6 hour

Reliability Growth



Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

More Reliability related Information

	Model	Time to achieve R = 0.9 for mission of length 900	Expected # of failures for next 900 time units	Nth failure	Expected times to next 1 failures
	All	All	All	All	All
1					
2	Geometric	275868.55873182	0.417554030265848	1	2170.03088926781

Ref: Fiondella, L. (2019). *Software Failure and Reliability Assessment Tool (SFRAT)*. UMASS Dartmouth. https://lfiondella.sites.umassd.edu/research/software-reliability/

Conclusion

- Software reliability poses unique challenges as compared to Hardware Reliability
- There are three phases in which reliability evaluation can be performed:
  - Development Phase -> Prediction Model (Metrics)
  - During Testing Phase -> Estimation Model (Failure Data)
  - Operation Life -> Estimation Model (Failure Data)
- Reliability prediction helps to predict reliability earlier to enable reliability enhancement by taking proactive measures and to avoid rework
- Reliability estimation gives actual reliability at a particular point of time
- Software Reliability is significant measure for evaluation of dependable software