Software Reliability

Anup Dewanji

Indian Statistical Institute, Kolkata

02 June 2018

Importance of Software

Almost all aspects of modern living require software. Some examples follow.

- Billing
- ATM
- On-line booking and purchase
- ECG monitoring in a hospital's ICU
- Auto-pilots flying an aircraft
- Banking
- Applications running on mobile phones

Importance of Software

Almost all aspects of modern living require software. Some examples follow.

- Billing
- ATM
- On-line booking and purchase
- ECG monitoring in a hospital's ICU
- Auto-pilots flying an aircraft
- Banking
- Applications running on mobile phones
- Preparation of this presentation

Software Failure

A software can be said to have a failure if it does not produce the intended result/output during its operation. This can happen due to many different reasons.

Software Failure

A software can be said to have a failure if it does not produce the intended result/output during its operation. This can happen due to many different reasons.

Famous software failures:

- The Y2K bug
- Mars Climate Orbiter by NASA in 1998
- Radiation overdose by Therac-25 machines in the 1980's
- Power blackouts in the US in 2003
- WannaCry ransomeware attack in 2017 on banking and other facilities

Software Reliability

It is important to have a measure of how reliable a software is.

Several definitions exist depending on the problem at hand.

A commonality among many definitions is that the measure should correlate with the probability of failure-free operation over a specified duration of usage.

Software Reliability

It is important to have a measure of how reliable a software is.

Several definitions exist depending on the problem at hand.

A commonality among many definitions is that the measure should correlate with the probability of failure-free operation over a specified duration of usage.

The reliability of a software is defined as the probability that the software will serve its intended purpose for a specified period of time (or, number of times) under a specified condition.

Major Objectives

- Identify relevant data and select/develop a modeling approach.
- Define reliability.
- Evaluate/compute reliability under the chosen model in terms of the unknown model parameters.
- Estimate the model parameters and hence the reliability.

Software Verification

Software verification has two parts:

(i) *static verification* ensures that the software meets the quality standards for coding, design and complexity, and algorithm, etc., and
(ii) *dynamic verification* ensures proper functioning of the software during its run-time.

This second part is also known as *Software Testing*, a process of executing the software program with the goal of finding errors/faults/defects, through application on test cases.

Software Testing

There are various software testing scenarios.

- Black-box Testing: The tester has no access to the source code and no knowledge of the software.
- White-box Testing: The tester has access to the source code and also knowledge of the software.
- Grey-box Testing: Some knowledge of the software, possibly no access to the source code.
- User-driven Testing: Users of a software voluntarily report a defect through internet with the defect being logged in a specialized database, called bug-database (e.g., beta-testing).

Software Testing

Once an error occurs leading to detection of a fault, it is debugged.
In this context, there are two aspects each with two scenarios:

1. Immediate debugging vs Periodic debugging

2. Perfect debugging vs Imperfect debugging

Software Reliability Models

The goal of software reliability modeling is to provide a probabilistic model for the software defect discovery process. There are several modeling approaches depending on the nature of the testing process and the available data.

Modeling is usually done on continuous time scale, measured through calender time, or CPU. One can also use a discrete time scale in which a single run of the software represents one unit of time.

Interestingly, modeling depends on the type of software testing process giving the nature of data that is to be expected, along with the associated limitations.

Software Reliability Models

1. Non-homogeneous Poisson Process (NHPP): Immediate Debugging

The cumulative defect counts $N(t)$ by time t follows a NHPP with a positive and non-decreasing mean value function $\Lambda(t) = \int_0^t \lambda(u)du$, where $\lambda(u)$ is interpreted as the defect detection rate, or intensity, at time t.

This implies

- The defect count $N(t)$ has a *Poisson* distribution with mean $\Lambda(t)$.
- For any time interval I, the number N_I of defect counts in I has a *Poisson* distribution with mean $\int_I \lambda(t)dt$.
- The defect counts N_I and N_J, in two disjoint time intervals I and J, are independent of each other.

Software Reliability Models

The intensity function $\lambda(t)$, or the mean function $\Lambda(t)$, is usually modeled as a parametric function and the model parameters are estimated by using the Poisson distribution of counts.

Different NHPP models:

(i) Goel-Okumoto (1979): $\lambda(t) = a\exp(-bt),\ a > 0, b > 0.$

(ii) Musa-Okumoto (1984): $\Lambda(t) = \frac{1}{\theta}\log(1 + \lambda_0 t),\ \lambda_0 > 0, \theta > 0.$

(iii) Ohba (1984): $\Lambda(t) = \frac{a}{1+\exp(-bt)},\ a > 0, b > 0.$

Software Reliability Models

2. Inter-Failure Time Models: Immediate Debugging

If S_i denotes the ith ordered detection time of a fault, then $T_i = S_i - S_{i-1}$ for $i = 2, \cdots, n$, with $T_1 = S_1$, are called the inter-failure times.

This approach models the joint distribution of $(T_1, \cdots, T_n)$ if n faults are detected.

The model parameters are estimated by using the joint distribution of n and the T_i's.

Software Reliability Models

Different models for the T_i's:

(i) Jelinski-Moranda (1972): T_i's are independent with T_i having an *Exponential*$(\lambda(\nu - i + 1))$ distribution.
(ii) Littlewood-Verral (1973): Same as (i) with the *Exponential* parameters having stochastically decreasing Gamma priors.
(iii) Moranda (1975): Same as (i) with the ith *Exponential* parameter being $\exp(\alpha + \beta i)$.
(iv): Singpurwalla and Soyer (1985): $T_i = \alpha_i T_{i-1}^{\theta_i}$ with the α_i's modeled as independent Log-Normal variates and the θ_i's modeled as an AR process.

Reliability of a Unit of Flight Control Software

Four versions corresponding to four testing activities with runs of Integrated Simulation Testing (IST). The schedule for the number of IST runs in each activity is as follows.

There are $N = 126$ modules in the software. The schedule is same for all the modules except two.

Version 1: 187 IST runs
Version 2: 227 IST runs
Version 3: 187 IST runs
Version 4: 267 IST runs

Reliability of a Unit of Flight Control Software

Four versions corresponding to four testing activities with runs of Integrated Simulation Testing (IST). The schedule for the number of IST runs in each activity is as follows.

There are $N = 126$ modules in the software. The schedule is same for all the modules except two.

Version 1: 187 IST runs
Version 2: 227 IST runs
Version 3: 187 IST runs
Version 4: 267 IST runs

Errors detected within an activity are corrected at the end of the activity.

Reliability of a Unit of Flight Control Software

Four versions corresponding to four testing activities with runs of Integrated Simulation Testing (IST). The schedule for the number of IST runs in each activity is as follows.

There are $N = 126$ modules in the software. The schedule is same for all the modules except two.

Version 1: 187 IST runs
Version 2: 227 IST runs
Version 3: 187 IST runs
Version 4: 267 IST runs

Errors detected within an activity are corrected at the end of the activity.

For each module, we observe the number of simulation runs and number of failures (runs with error) in different IST activities. There are altogether six failures (detected errors), but no module having more than one failure.

Reliability of a Unit of Flight Control Software

Definition of Reliability in this case:

Given all the design changes (that is, detection and correction of errors, modification of the software to accommodate new requirements, etc.) up to the current time, we define the reliability of the unit of software as the probability of no failure due to this particular software unit, if another single run of the Integrated Simulation Testing is to be carried out at this time with a randomly selected input from the set of all possible inputs under the identical computing environment and other conditions.

Note that this **definition of reliability** depends on the current time, in particular, the experience up to the current time. This is eventually to be equated with the probability of no failure in this particular software unit in a mission.

Reliability of a Unit of Flight Control Software

Some Features of the Data:

- The time domain is inherently discrete with each run as one discrete unit of time.
- Debugging is not immediate.
- There is a modular structure.
- Identity of the module containing the error that caused failure is available.
- There are module-specific covariates (e.g., complexity, history of errors during Code Inspection and Module Testing).
- Different modules have different extent of exposure to testing.
- Identity of the failed run is not available.

Reliability of a Unit of Flight Control Software

Modeling:

A framework for fitting a Binomial distribution is very apparent from earlier description.

Since the detected errors are corrected only at the end of one IST activity, the failure probability of a particular module may be assumed to be the same over the runs within an activity.

This probability of a failure (run with error) or success (error-free run) can be modeled suitably to incorporate regression variables (covariates) in the spirit of Generalized Linear Models, in particular, through the use of logistic regression model.

Reliability of a Unit of Flight Control Software

Reliability of this unit of software, after the testing process in the four activities are over, is computed as the product, over the 126 modules, of the conditional probability, given the experience up to the end of testing, of an error-free run, assuming the modules to act independently.

This reliability will involve the associated model parameters which need to be estimated from the observed data.

The model parameters can be estimated through the maximum likelihood method along with its standard error.

Reliability of a Unit of Flight Control Software

Reliability of this unit of software, after the testing process in the four activities are over, is computed as the product, over the 126 modules, of the conditional probability, given the experience up to the end of testing, of an error-free run, assuming the modules to act independently.

This reliability will involve the associated model parameters which need to be estimated from the observed data.

The model parameters can be estimated through the maximum likelihood method along with its standard error.

Another measure of reliability, similar to Mean Time to Failure (MTTF), is the *expected number of simulation runs needed to have the next error-producing run, given the current testing experience*.

Reliability of a Unit of Flight Control Software

The estimated reliability is 0.9979 with standard error 0.0026. The approximate 95% confidence interval is obtained as [0.9929,1.0030].

Reliability of a Unit of Flight Control Software

The estimated reliability is 0.9979 with standard error 0.0026. The approximate 95% confidence interval is obtained as [0.9929,1.0030].

The estimated MTTF is 476.19 with standard error 1.2381. The approximate 95% confidence interval is obtained as [473.76,478.62].

The Modern Bug Database: Analysis of Python Data

- Specialized software that enables software users to report defects through the Internet
- Reported defects are reviewed by software engineers to determine whether they are really defects or not
- Every confirmed and distinct defect carries a record in the database containing
  - The calendar date when the defect was first reported
  - The type of defect (Example: Security, Performance, Behavior, Crash)
  - The software component affected by the defect
- Example of a bug database: http://bugs.python.org

The Modern Bug Database: Analysis of Python Data

Challenges in Bug Databases for Software Reliability Modeling:

- Contain user-reported software defects subject to uncontrolled and unavailable usage of the software
- Usage rate is a function of time and is unknown
- Reporting depends on the user (specialist vs casual)
- Distribution of defect type depends on unknown use cases of the software
- No existing model for software reliability considers these important confounding factors
- Due to uncertain usage pattern, modeling based on time scale is inappropriate

The Modern Bug Database: Analysis of Python Data

A Novel Modeling Approach:

- Uses the classification of reported defects into multiple types out of which one is of primary interest for reliability purpose
- Model is nonparametric w.r.t. the uncertain usage rate leading to a partial likelihood analysis
- Reliability metrics do not depend on the usage rate

The partial likelihood is developed in terms of, for each reported case, the conditional probability that the currently reported defect is of the recorded type, given the types and times of the previously reported cases and the time of the currently reported case. This does not depend on the usage pattern.

Model parameters are estimated via GLM procedures.

The Modern Bug Database: Analysis of Python Data

Snapshot of the Python Bug Database as on Jan 31 2012:

Table : Summary of defects in the two versions of Python

	Python 2.7		Python 2.6	
Defect Type	Count	Percentage	Count	Percentage
Crash	130	5.7 %	124	6.3 %
Security	19	0.8 %	16	0.8 %
Others	2124	93.5 %	1835	92.9 %
Total Bugs	2273	100%	1975	100%

The Modern Bug Database: Analysis of Python Data

Reliability Metrics: Assume only two types of defects; for the Python data, these can be Crash and Others, Crash being the primary type.

Mean Number of Defects to Failure (MNDF): The expected number of defects other than the Crash type to be reported before a Crash type is reported.

Reliability $R(N)$: Probability of not having a Crash type defect reported in the next N reported defects.

Both the metrics do not depend on the underlying usage rate, but on the other model parameters.

The Modern Bug Database: Analysis of Python Data

Reliability Metrics: Assume only two types of defects; for the Python data, these can be Crash and Others, Crash being the primary type.

Mean Number of Defects to Failure (MNDF): The expected number of defects other than the Crash type to be reported before a Crash type is reported.

Reliability $R(N)$: Probability of not having a Crash type defect reported in the next N reported defects.

Both the metrics do not depend on the underlying usage rate, but on the other model parameters.

These metrics can be used for the three-type analysis as well.

The Modern Bug Database: Analysis of Python Data

Table : The estimates of the reliability metrics

		Python 2.7		Python 2.6	
Analysis	Metrics	Estimate	SE	Estimate	SE
Three-type	$R(N = 10)$(Crash)	0.43	0.10	0.49	0.13
	$MNDF$(Crash)	12.02	2.67	15.06	5.50
	$R(N = 10)$(Security)	0.98	0.02	0.90	0.10
	$MNDF$(Security)	55.91	17.67	55.15	20.28
Two-type	$R(N = 10)$(Crash)	0.44	0.07	0.47	0.07
	$MNDF$(Crash)	11.23	2.09	12.28	2.26

THANK YOU