An Initiative to Improve Quality of Software Testing Techniques and Calculating Total Number of Failures using Bayesian Method

Pranoy Dev (16BEC0098)

***Abstract*- The current concerns relating to the quality of analysis performed in existing studies reveals the necessity for ways and tools to help the definition and execution of empirical studies and experiments. This paper discusses those problems specific to the analysis of software testing techniques and proposes an initiative for a cooperative effort to encourage duplicability of experiments evaluating software testing techniques. Total number of failures of a software system can help testers to have a better understanding of the software quality. This paper proposes a model to predict the total number of software failures in a system by analysing the failure data from testing. Using the failure data and the code coverage, and combining them in a Bayesian way, we will approach towards the result. The methodology is applied to real world failure data to validate its predictability. The predictive accuracy of our model is also evaluated with different test cases. The results of our experiment shows that our proposed model can provide a very good estimation of the total number of failures. The estimation is stable from a very early point on.**

I. INTRODUCTION

Testing is defined as a process of evaluation that either the specific system meets its originally specified requirements or not. It is mainly a process encompassing validation and verification process that whether the developed system meets the requirements defined by user. Therefore, this activity results in a difference between actual and expected result. Software testing refers to finding bugs, errors or missing requirements in the developed system or software. So, this is an investigation that provides the stakeholders with the exact knowledge about the quality of the product. Software Testing can also be considered as a riskbased activity.

II. LITERATURE SURVEY

A. EXISTING TESTING METHODS

For the commencement of the testing process, the primary step is to generate test cases. The test cases are developed using numerous testing techniques, for the effective and accurate testing. The main testing techniques are Black Box testing, White Box testing and Grey Box testing. White Box testing is considerably effective because it is the technique of testing that not only tests the practicality of the software but additionally tests the internal structure of the application. While designing the test cases to conduct white box testing, programming skills are requisite. White box testing is additionally referred to as clear box or glass box testing. This type of testing are often applied to all or any levels as well as unit, integration or system testing. This sort of testing is also referred to as Security Testing that it fulfils the necessity to work out whether or not the information systems protect data and maintains the intended functionality. As this type of testing method makes use of the internal logical arrangement of the software thus it's capable enough of testing all the independent ways of a module, each logical decision is exercised, all loops are checked at every boundary level, and internal data structures are also exercised. However, white box testing serves a purpose for being a complex testing method due to the inclusion of programming skills within the testing process. Black box testing is a testing technique that essentially tests the functionality of the application without going into its implementation level detail. This method are often applied to each level of testing within the SDLC. It mainly executes the testing in such some way that it covers each and every functionality of the application to determine whether it meets the initially specified requirements of the user or not. It’s capable of finding incorrect functionalities by testing their functionality at every minimum, maxi mum and base case value. It’s the most simple and widespread testing method used worldwide.

B. ISSUES

The empirical software engineering community has been dedicating efforts in breaking down the fundamental elements of an experiment in software engineering. Despite the efforts and discussions regarding appropriate methods to conduct experiments in software engineering, evaluating STT is not easy since many information and artefacts are required in order to thoroughly evaluate the capabilities of a testing technique. First of all, software testing itself is a very multidisciplinary area that is able to comprise distinguished and combined types of software such as standalone, web services, real time, critical systems, embedded systems, etc. Therefore, drawing a precise line to establish the scope affecting the test is very challenging. At the same time, trying to draw that line is one of the most important steps to define a good experiment, since it allows researchers to identify: context, subjects, limitations, and more importantly, which variables can/must be controlled. Why are we not seeing, over the years, a significant increase in the number of empirical findings regarding STT? There is an array of classifications, configurations and guidelines that help researchers to define, conduct and report their experiments. But manifesting those methods for constructs specific to STT evaluation is very challenging. For example, statistics often require large sets of significant data (usually hard to obtain or unavailable) to achieve conclusive results. In addition, the need for human involvement in testing makes the generation of a large number of test sets infeasible. Similarly, obtaining defect data for larger test sets is nearly impossible since detection of defects will always be, to a certain degree, a stochastic process. Availability and access to information is still limited when evaluating STT. For instance, the System under Test (SUT) may not be ready for testing, test cases may present limited coverage, and defect data is unknown, among others. Even when available, many researchers disregard the representativeness of their artefacts (e.g. choice of inappropriate programs or unreal defects) or even the minimum sample size required to claim statistical significance of results. That creates gaps and validity threats that seriously compromise credibility of results. There are existing techniques for stochastic generation of data for testing. More specifically, they rely on models to generate well-formed data. Those types of strategies enable control over automatic generation of large sets of artefacts, hence assisting experimenters who struggle with availability and accessibility of objects in an experiment. Thus, existing approaches can already be applied to overcome those issues. Besides getting data, organization and analysis of all elements in an experiment intimidates many researchers. Devising an appropriate experimental design to comply with one’s hypotheses, objectives and variables is overwhelming and a poor design lead to confusing reports. Similarly, lack of experience and/or knowledge in statistics hinder researchers to harness full potential of their data causing them to either enhance or destroy credibility of conclusions. For example, many researchers recklessly use parametric tests (such as ANOVA or t-test) as a rule of thumb, without proper investigation of its assumptions, sometimes leading to erroneous conclusion. In its early start, the empirical software engineering community turned to other disciplines that have been dealing with empirical evaluation over decades (e.g. biology and social sciences) in order to overcome general difficulties in managing data, subjects, statistical analysis, etc. The software testing community seems to be performing more experiments. However, they neglect validation of existing experiments and proposal of innovative strategies to help their fellow researchers in overcoming the specific challenges of evaluating STT. That being said, validation of experiments needs to be performed through reproduction, replication or re-analysis of existing experiments with STT.

III. METHODOLOGY

A. MODEL-BASED TESTING

Model-based testing is a software testing technique in which the test cases are derived from a model that describes the functional aspects of the system under test. Generally, it makes use of a model to generate tests that includes both offline and online testing. Some advantages of model-based testing are:

a) Higher level of Automation is achieved.

b) Exhaustive testing is possible.

c) Changes to the model can be easily tested.

B. GHOST TRANSITION TESTING

These testing for part of model based testing wherein the only difference that lies between ghost transition testing and all pair testing is that the transitions which are not defined/ visible for a state (called as ghost transition) when triggered will remain in same state as the original. Such type of transitions when triggered does not affect the change in state.

C. MULTIPLE CONDITION TESTING

Development of tests by white box testing method in which test cases are designed to execute combinations of single condition outcomes and each of the conditions or branches are evaluated.

D. NUMBER OF FAILURES

It is highly plausible, that the additional information of code coverage could help us to predict the total number of failures better than with failure data alone. Here for the Bayesian computations the hyper geometric distribution is used as a prior. This method to use the hyper geometric distribution is known and used as the "Capture-Recapture Method" in biology since many decades. It goes back to P.S. Laplace. He used it in the year 1786 to estimate the population size of France. Here we use it to estimate the total number of failures. This distribution can be used if we know that the faults (the causes of our failures) are somehow spread “uniformly” all over the code. To verify this assumption we check the relationship of code coverage achieved and number of detected failures. In the data set examined in the next chapter there is an approximately linear relationship between coverage achieved and failures detected. Other people observed this relationship too (at least after some initial time).

IV. IMPLEMENTATION

For our computations we need as input:

a) number of failures detected during the test

b) total number of statements (or whatever is used for coverage)

c) percentage of coverage reached

$$p= \frac{\binom{s}{n}\cdot\binom{S-s}{N_0-n}}{\binom{S}{N_0}}$$

where
N0=Total number of failures n= number of failures detected S= total number of statements
s= number of statements covered

V. RESULTS

A. CODE

```
#include<stdio.h>
#include<conio.h>
#include<iostream> using
namespace std;
class VendingMachine {
 public:
          VendingMachine() ;
int coin();       int small_cup()
;          int large_cup();
int sugar();
             int tea();
          int insert_large_cups(int);
int insert_small_cups(int);     int
set_price(int p);          int cancel() ;
int dispose() ;
             void showvariable();


private:
int x; int
price; int


k; int k1;
int t; int
s;
```

```
};

VendingMachine::VendingMachine() {
          k1=0;
k=0;     t=0;
          price=0;
x=1;
}


int VendingMachine::coin() {
          if (x==1) {
if
((t+25>=price)&&(price>0)) {
                    s=0;
t=0;                          x=2;
                                  return 1;
                         }
                         else if(t+25<price) {
                              t=t+25;
          return 1;
                         }
             }
             else if ((x>1)&&(x<6)) {
                         std::cout<<"RETURN
COIN"<<std::endl;
                         return 1;
             }
             return 0;
}


int VendingMachine::small_cup() {
             if ((x==2)||(x==3)) {
                         s=2;
                         return 1;
             }
             return 0;
}

int VendingMachine::large_cup() {
if ((x==2)||(x==3)) {

                         s=1;
                         return 1;
             }
             return 0;
```

```
}

int VendingMachine::sugar() {
if ((x==2)||(x==3)) {
                if(x==2) x=3;
else x=2;
                    return 1;
          }
          return 0;
}


int VendingMachine::tea() {
          if ((x==2)||(x==3)) {

           if((x==2)&&(k1>1)&&(s==2)) {

          std::cout<<"DISPOSE SMALL
CUP OF TEA"<<std::endl;
                        k1=k1-1;
        x=1;
                              return 1;
                    }
                    else if
((x==2)&&(k>1)&&(s==1)) {

          std::cout<<"DISPOSE LARGE
CUP OF TEA"<<std::endl;
                              k=k-1;
                              x=1;
                              return 1;
                    }
                    else if
((x==2)&&(k==1)&&(s==1)) {

          std::cout<<"DISPOSE LARGE
CUP OF TEA"<<std::endl;
                              k=k-1;
                              x=5;
                              return 1;
                    }
                    else if
((x==2)&&(k1==1)&&(s==2)) {

          std::cout<<"DISPOSE SMALL
CUP OF TEA"<<std::endl;
                              k1=k1-1;
                              x=4;
                              return 1;
                    }
                    else if
((x==3)&&(k1==1)&&(s==2)) {

          std::cout<<"DISPOSE SMALL
CUP OF TEA WITH
SUGAR"<<std::endl;
                        k1=k1-1;
        x=4;
                              return 1;
                    }
                    else if
((x==3)&&(k==1)&&(s==1)) {

          std::cout<<"DISPOSE LARGE
CUP OF TEA WITH
SUGAR"<<std::endl;
                        k=k-1;
        x=5;
                              return 1;
                    }

           if((x==3)&&(k1>1)&&(s==2)) {

          std::cout<<"DISPOSE SMALL
CUP OF TEA WITH
SUGAR"<<std::endl;
                        k1=k1-1;
        x=1;
                              return 1;
                    }
                    else if
((x==3)&&(k>1)&&(s==1)) {

          std::cout<<"DISPOSE LARGE
CUP OF TEA WITH
SUGAR"<<std::endl;
                              k=k-1;
                              x=1;
                              return 1;
                    }
                    return 0;
          }
          return 0;
```

```
}

int VendingMachine::insert_large_cups(int
n) {
            if ((x==1)&&(n>0)) {
                   k=k+n;
return 1;
            }
            else if ((x==5)&&(n>0)) {
                   k=n;
x=1;
                        return 1;
            }
            return 0;
}

int
VendingMachine::insert_small_cups(int n)
{
            if ((x==1)&&(n>0)) {
                   k1=k1+n;
return 1;
            }
            else if ((x==4)&&(n>0)) {
                   k1=n;
x=1;
                        return 1;
            }
            return 0;
}

int VendingMachine::set_price(int p) {
            if ((x==1)&&(p>0)) {
                   price=p;
return 1;
            }
            return 0;
}


int VendingMachine::cancel() {
if ((x==2)||(x==3)) {
                        std::cout<<"RETURN
COINS"<<std::endl;
                        x=1;
                        return 1;
            }
            return 0;
}

int VendingMachine::dispose() {
            if ((x==1)) {
                        std::cout<<"SHUT
DOWN"<<std::endl;
                        x=6;
                        return 1;
            }
            return 0;
}


void VendingMachine::showvariable()
{
std::cout<<"x ="<< VendingMachine::x
<<"\n" ; std::cout<<"k =" << k
<<"\n"; std::cout<<"k1 =" <<
k1<<"\n"; std::cout<<"t =" << t
<<"\n"; std::cout<<"price =" << price
<<"\n" ; std::cout<<"s =" << s <<"\n";
} int fact(int n){     if(n==0)
return 1;     if (n>0) return
n*fact(n-1);
};

int NCR(int n,int r){     if(n==r)
return 1;     if (r==0&&n!=0)
return 1;     else return (n*fact(n-
1))/fact(n-
1)*fact(n-r);
};

 int main()  {    int
i,n,p,p1,n1,s1,S,N0;
  char ch,op;
  VendingMachine vm;
do
  {
  std::cout<<" \t         -----VENDING
MACHINE MENU -----        "<<"\n";
  std::cout<<"----------------------------------
---"<<"\n";
  std::cout<<"|| \t  1. Coin inserted
"<<"\n";    std::cout<<"|| \t  2. Small
cup()"<<"\n";    std::cout<<"|| \t  3. Large
cup()"<<"\n";    std::cout<<"|| \t  4.
Sugar() "<<"\n";    std::cout<<"|| \t  5.
Tea()"<<"\n";
   std::cout<<"|| \t  6. Insert small
cup()"<<"\n";    std::cout<<"|| \t
7. Insert large cup()"<<"\n";
   std::cout<<"|| \t  8. Set Price"<<"\n";
std::cout<<"|| \t  9. Dispose"<<"\n";
std::cout<<"|| \t  f. Failures"<<"\n";
std::cout<<"|| \t  0. Cancel()"<<"\n";
std::cout<<"|| \t  s. Show variables()"
<<"\n";
   std::cout<<"---------------------------------
-----"<<"\n";    std::cout<<"Please enter
your choice: ";    std::cin>>ch;
switch(ch)    { case '1': n= vm.coin();
                              std::cout<<" The value
returned by method is:"<<n <<"\n";
                              break;

                case '2':
                        n=vm.small_cup();
std::cout<<" The value
returned by method is:"<<n<<"\n";
                              break;

     case '3': n=vm.large_cup();
                              std::cout<<" The value
returned by method is:"<<n<<"\n";
                              break;

     case '4':  n=vm.sugar();
                              std::cout<<" The value
returned by method is:"<<n<<"\n";
                              break;

                case '5' : n=vm.tea();
                        std::cout<<" The value
returned by method is:"<<n<<"\n";
break;

  case '6':    std::cout<<" Enter the no of
small cups:";
                              std::cin>>i;
                              n =
vm.insert_small_cups(i);
                        std::cout<<" The value
returned by method is:"<<n<<"\n";
break;

             case '7': std::cout<<" Enter the no
of large cups: ";
                        std::cin>>i;
               n=vm.insert_large_cups(i);
                              std::cout<<" The value
returned by method is:"<<n<<"\n";
                              break;

                case '8': std::cout<<"Enter the
price to be set: ";
                        std::cin>>p;
n=vm.set_price(p);                std::cout<<"
The value
returned by method is:"<<n <<"\n";
                              break;

             case 's': std::cout<<"\n";
vm.showvariable();
                  break;

                case '0': n= vm.cancel();
                              std::cout<<" The value
returned by method is:"<<n <<"\n";
                        break;      case
'9':
                  n = vm.dispose();
                              std::cout<<" The value
returned by method is:"<<n <<"\n";
                        break;   case 'f':
        cout<<"Enter total no. of failures";
cin>>N0;        cout<<"Enter no. of
failures detected";        cin>>n1;
cout<<"No. of statements covered";
cin>>s1;        cout<<"Total No. of
statements
covered";
cin>>S;
        cout<<"Posterior distribution of total
no. of failures";
p1=(NCR(s1,n1)*NCR(S-
s1,N0n1))/NCR(S,N0);
        cout<<p1;
        break;
   default: std::cout<<"\n"<<"Please enter
valid option... !!!!"<<"\n";

}
  std::cout<<"\n Press (y/n) to continue :"
<<"\t";
  std::cin>>op;
}while(op=='y');

return 0;
}
```

B. TEST CASES

Test#1: insert_large_cups 5 insert_small_cups 5 set_price 25 coin small_cup coin tea coin cancel dispose

Test#2: insert_large_cups 1 insert_large_cups 1 insert_small_cups 1 insert_small_cups 1 set_price 25 coin large_cup tea coin large_cup coin tea coin insert_large_cups 1 coin dispose

Test#3: insert_large_cups 1 set_price 50 insert_large_cups 1 coin insert_large_cups 1 insert_small_cups 1 coin dispose

Test#4: insert_large_cups 1 insert_small_cups 1 set_price 30 coin coin large_cup sugar large_cup tea coin insert_large_cups 1 dispose

Test#5: insert_large_cups 2 set_price 50 coin coin tea insert_large_cups 1 large_cup tea insert_large_cups 1 coin coin cancel dispose

Test#6: insert_large_cups 2 set_price 24 coin coin large_cup coin sugar large_cup coin large_cup tea coin coin insert_large_cups 1 dispose

Test#7: insert_large_cups 2 set_price 26 coin coin tea dispose large_cup tea dispose

Test#8: insert_small_cups 2 set_price 35 coin coin coin tea small_cup tea dispose

Test#9: insert_small_cups 2 set_price 50 coin coin cancel coin coin small_cup tea insert_small_cups 1 insert_small_cups 1 coin coin sugar sugar sugar small_cup coin tea insert_small_cups 1 coin dispose
Test#10: insert_small_cups 1 set_price 45 coin coin small_cup tea coin insert_small_cups 1 set_price 10 dispose

Test#11: insert_small_cups 1 insert_large_cups 1 set_price 50 coin coin large_cup sugar tea insert_large_cups 1 coin coin set_price 40 coin sugar small_cup tea coin coin insert_small_cups 1 insert_small_cups 1 dispose

Test#12: insert_large_cups 1 insert_small_cups 1 set_price 45 coin coin large_cup small_cup small_cup sugar coin small_cup large_cup small_cup tea insert_large_cups 1 insert_small_cups 1 insert_large_cups 2 dispose

Test#13: set_price 50 coin insert_large_cups 1 insert_small_cups 1 coin small_cup large_cup large_cup coin large_cup tea insert_large_cups 1 insert_small_cups 1 coin sugar tea coin sugar small_cup tea dispose

Test#14: coin set_price 50 coin insert_small_cups 2 coin coin coin tea small_cup tea insert_small_cups 1 coin coin tea small_cup tea insert_small_cups 1 insert_small_cups 1 insert_large_cups 1 set_price 25 set_price 35 insert_small_cups 1 coin coin cancel set_price -1 dispose

Test#15: dispose insert_small_cups 1 set_price 20 insert_small_cups 1 coin insert_small_cups 1 insert_small_cups 1 coin dispose

Test#16: coin insert_small_cups 5 coin set_price 50 coin coin small_cup cancel coin coin small_cup tea insert_large_cups 1 coin coin large_cup small_cup tea dispose

Test#17: coin insert_large_cups 2 set_price 29 coin coin set_price 30 cancel set_price 33 insert_large_cups 2 coin coin sugar cancel set_price 30 coin coin sugar tea large_cup tea dispose

Test#18: coin coin insert_large_cups 3 insert_small_cups 2 set_price 27 coin coin tea tea tea insert_large_cups 1 insert_small_cups 1 set_price 20 large_cup small_cup cancel coin coin tea tea large_cup cancel insert_small_cups 1 set_price 30 coin coin large_cup small_cup tea insert_small_cups 1 coin coin large_cup tea small_cup tea insert_small_cups 1 coin coin coin cancel insert_large_cups -1 insert_large_cups 1 coin coin coin small_cup coin tea coin dispose

Test#19: insert_small_cups 1 insert_large_cups 2 coin set_price 30 coin coin large_cup coin small_cup large_cup tea insert_large_cups 1 coin coin sugar sugar tea small_cup tea insert_small_cups 1 coin coin sugar sugar tea large_cup tea
coin insert_large_cups 1 dispose

Test#20: coin coin insert_small_cups 2 set_price 30 coin coin insert_small_cups 1 cancel insert_large_cups 2 coin coin sugar small_cup large_cup large_cup small_cup small_cup tea dispose

Test#21: set_price 30 insert_large_cups 2 coin coin small_cup sugar sugar cancel coin insert_large_cups 1 coin large_cup small_cup tea large_cup tea insert_large_cups -1 insert_large_cups 2 dispose

Test#22: set_price 50 coin insert_small_cups 2 insert_large_cups 3 coin large_cup coin tea insert_large_cups 1 set_price 28 coin coin tea small_cup tea set_price 20 insert_small_cups 2 coin small_cup tea coin dispose

Test#23: insert_large_cups 3 set_price 30 coin coin large_cup sugar coin tea insert_small_cups 1 coin small_cup coin sugar small_cup large_cup tea coin sugar coin coin cancel dispose

Test#24: insert_small_cups 2 set_price 30 coin coin small_cup coin sugar sugar tea coin tea insert_large_cups 2 insert_small_cups 1 coin small_cup coin sugar coin tea insert_small_cups 1 dispose

Test#25: insert_small_cups 2 insert_large_cups 2 set_price 30 coin coin large_cup coin sugar sugar tea insert_large_cups 2 dispose

Test#26: insert_large_cups -1 insert_large_cups 2 set_price 50 coin coin large_cup coin sugar large_cup coin tea insert_large_cups 2 dispose

Test#27: insert_large_cups 2 insert_small_cups 2 set_price 40 coin coin sugar sugar coin sugar coin sugar large_cup sugar sugar small_cup sugar small_cup tea insert_large_cups 0 dispose

Test#28: insert_small_cups 2 set_price 30 coin coin small_cup coin sugar small_cup cancel coin coin sugar small_cup sugar sugar coin small_cup tea insert_small_cups -1 dispose

Test#29: insert_large_cups 2 insert_small_cups 2 set_price 35 coin coin small_cup large_cup coin sugar large_cup cancel insert_small_cups 2 coin coin sugar large_cup sugar sugar large_cup tea insert_large_cups -2 dispose

Test#30: insert_large_cups 2 insert_small_cups 2 set_price 40 coin coin sugar small_cup large_cup tea coin sugar tea insert_large_cups 1 dispose Test#31: set_price 60 insert_large_cups 5 coin coin coin large_cup sugar coin tea set_price 0 dispose

Test#32: set_price -1 set_price 33 insert_large_cups 1 coin coin coin large_cup sugar coin tea coin dispose

Test#33: set_price 40 coin coin coin sugar cancel insert_large_cups 2 coin coin large_cup sugar cancel coin dispose Test#34: insert_large_cups 2 set_price 50 coin coin large_cup coin sugar large_cup tea set_price 5 dispose

Test#35: insert_large_cups 1 set_price 24 coin coin large_cup coin sugar large_cup tea coin dispose

Test#36: insert_large_cups 2 set_price 29 coin coin coin large_cup tea set_price 10 dispose

Test#37: insert_small_cups 0 insert_large_cups 0 set_price -1 cancel coin coin sugar tea small_cup large_cup set_price 29 coin dispose

Test#38: insert_small_cups 1 insert_large_cups 3 set_price 40 coin coin dispose insert_small_cups 2 insert_large_cups 1 set_price 25 cancel dispose

Test#39: insert_small_cups 1 insert_large_cups 1 set_price 25 coin coin sugar insert_small_cups 2 insert_large_cups 1 set_price 30 dispose small_cup tea insert_small_cups 1 dispose

Test#40: insert_large_cups 1 set_price 30 coin coin sugar large_cup tea small_cup large_cup set_price 40 cancel sugar tea dispose insert_small_cups 2 insert_large_cups 2 dispose

Test#41: insert_small_cups 1 set_price 30 coin coin sugar small_cup tea small_cup large_cup set_price 40 cancel sugar tea dispose insert_large_cups 2 insert_small_cups 2 dispose

Test#42: insert_small_cups 1 set_price 35 coin coin sugar coin tea small_cup tea dispose

Test#43: insert_small_cups 2 set_price 26 coin coin small_cup sugar tea dispose Test#44: insert_large_cups 1 insert_small_cups 1 set_price 30 coin coin small_cup coin tea insert_small_cups 1 dispose

Test#45: insert_small_cups 1 set_price 26 coin coin small_cup sugar sugar tea insert_small_cups 1 insert_large_cups 1 coin coin large_cup sugar sugar tea insert_large_cups 1 dispose

VI. FUTURE WORK

The initiative begins with a collaborative effort within the testing community to share and standardize methods and artefacts used to evaluate STT. Therefore, the research comprises the state of art of empirical studies in software engineering, software testing, statistics, and experimental designs, among others. The outcome is the development of techniques and tools to provide support and execution of experiments. Therefore, researchers will extend the body of knowledge by proposing new methodologies to perform reproducible studies with STT. Even though replication and re-analysis are just as important as reproducibility, we believe that focusing first in reproducibility will allow researchers to quickly overcome the general lack of availability and

accessibility of data required to replicate/re-analyse the existing experiments. Thus, as the initiative strengthens, we enable more replication because the community will rely on a larger repository of experiments. In summary, we intend to encourage practices similar to the initiative of reproducible software engineering but adapting them to the software testing community. We begin by defining input and output of our process. The input are the study parameters, such as the objects of study, a
SUT, sets of test cases, number of subjects, a general hypothesis and goals. The output, in turn, is a compendium (i.e. a package), named Reproducible Software Testing Research Compendium (RSTRC).

VII. CONCLUSION

The proposed approach (Bayesian Zipfmodels+coverage) to predict the total number of failures of a software system can be used with failure time data or the number of test cases. In both cases it can also be used with grouped data. This approach gives excellent predictive performance with the data set of Wang et al.. The same methodology can be used to fit time or number of test cases vs. code coverage. In the future, we would like to extend our experiment to compare the estimated constant c in Zipf(c) with the shape of the usage profile (at a functional level), which can also be described via Zipf's law. It is conjectured, that there is a strong connection, which can be used for even better estimation at early levels (prior distribution for c from the usage profile). As the predicted mean value function fits the - in the future observed - failure data extremely well, the method described in and could be used very early to detect failure prone modules or irregularities during the test process (statistical process control).

VIII. REFERENCES

[1] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, Akhlaq Ahmad "Software Testing Techniques: A Literature Review"

[2] Jai Gaurl, Akshita Goya1, Tanupriya Choudhury and Sai Sabitha "A Walk Through of Software Testing Techniques"

[3] Francisco G. de Oliveira Neto, Richard Torkar, Patricia D. L. Machado "An Initiative to Improve Reproducibility and Empirical Evaluation of Software Testing Techniques"

[4] Harald A. Stieber, Linghuan Hu, W. Eric Wong" Estimation of the Total Number of Software Failures from Test Data and Code Coverage"

[5] Eduard P. Enoiu∗, Adnan ˇCauˇsevi´ ,Daniel Sundmark, Paul Petersson "A Controlled Experiment in Testing of Safety-Critical Embedded Software"

[6] S. Wang, Y. Wu, M. Lu, and H. Li, "Software Reliability modeling based on test coverage," in Proceedings of 9th International Conference on Reliability, Maintainability and Safety, Guiyang, China, 2011