FAILURE PREDICTION IN COMPLEX COMPUTER SYSTEMS

Paweł Janczarek

Abstract: *Failure prediction has its origins at the turn of the 15th and 16th centuries. Trade companies tried to calculate what was the risk with shipping goods over the sea. Modern failure predictions start in 70's of 20th century. Naturally software prediction has its origins in failure prediction of electronic equipments. Nomenclature and definitions are very often similar to those used in electronics. Since 70's scientists have been trying to find perfect model to simulate failures in software.*

Keywords: *system availability , failure, prediction, reliability, risk management*

ACM Classification Keywords: *G. Mathematics of Computing, G.3 PROBABILITY AND STATISTICS, G.3 PROBABILITY AND STATISTICS, Reliability and life testing*

Introduction

Failure prediction has its origins at the turn of the 15th and 16th centuries. Trade companies tried to calculate the risk related to shipping the goods over the sea. Modern failure predictions started in 70's of 20th century. Naturally software prediction has its origins in failure prediction of electronic equipment. Nomenclature and definitions are very often similar as those used in electronics. Since 70's scientists have been trying to find perfect model to simulate failures in software.

Failure prediction in software

At first we need to answer the following questions: *What is a failure in complex information system? Is this complete system shutdown? Is this delay in response for user?* Every system is very unique so for everyone of them failure has to be defined in slightly different manner.

Second very important issue is how we define the measure of time, or more precisely what type of clock we use to measure time between the failures. Is this human-clock time (days, hours, seconds, etc.) or we use hardware-clock time (CPU, logical executions, etc). Right time measure is crucial for software failure prediction. Natural measure for people is normal human-clock, but in context of software it is very often misleading. Systems often have their own life-cycle (batch processing, night recalculations, backups, etc.) not very correlated with human-clock way of perceiving the time aspect.

We also need to know what are the origins of our failures. Are they human errors, bugs in software, hardware malfunctions or else. According to Gartner analysis 80 % of errors are those in direct cause of human mistake. In our future analysis it is good to think if we would like to divide them separately.

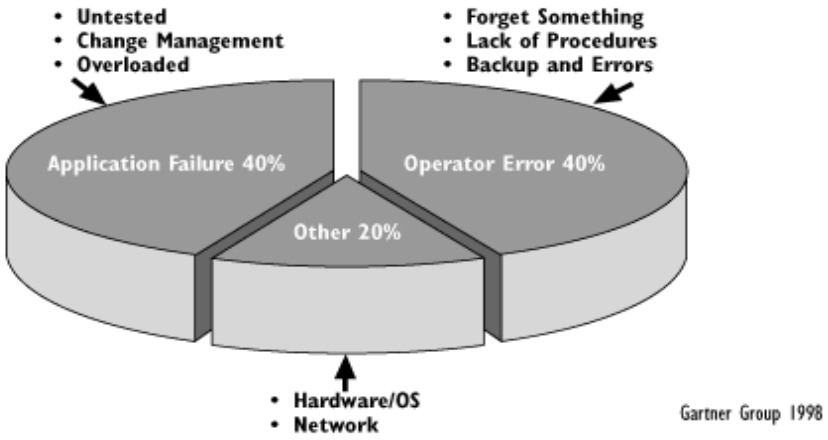

Figure 1. Origins of software errors

Basic theoretical background in failure prediction

Theory used for software failure predictions has its origins in hardware failure prediction. That's way names and formulas are very similar to those used for hardware (or sometimes even form theirs predecessor - risk calculation for shipping goods over the sea from the turn of the 15th and 16th centuries).

We can compute probability of a failure in very simple way as [Summerville]:

$$P(T \le t_0) = \int_0^{t_0} f_T(t)dt \tag{1}$$

where $f_T(t)$ – is the density function which describes distribution of failures in time. We can also calculate probability of failure in case there hasn't been any error for some time (Z(t) Δt). To do this we need to use conditional probability [Summerville]:

$$Z(t)\Delta t = P(t < T < t + \Delta t | T > t) \tag{2}$$

where T is time of failure, and Δt is time period in which failure will occur.

As we see to do all those calculations we need to have density function which describes distribution of failures in time. Without this density function identified we are unable to do any predictions of failures. Using empirical distribution functions based on history of previous failures we are able to fairly approximate distribution of failures for software. Simple example is presented in the next section.

At this moment we can calculate reliability of our system, which is telling us that it won't break down before t_0 point in time, with the following formula [Summerville]:

$$R_T(t_0) = P(T \ge t_0) = 1 - F_T(t_0) = 1 - \int_0^{t_0} f_T(t)dt = \int_{t_0}^{\infty} f_T(t)dt \tag{3}$$

The least value, the better reliability is between two systems. As one can see, the most important is the density function $f_T(t)$. Without it we can't do any predictions.

In practical cases, very often we can assume that Δt is very short so we can pass it over. Then we can simplify our conditional probability of failure to following equation [Summerville] (aka failure rate):

$$Z(t) = \frac{f_T(t)}{R_T(t)} \tag{4}$$

In almost all bibliography and electronics we can very often run across with term of MTTF (Mean Time To Failure). Using our previous calculation, we can now calculate MTTF, as:

$$MTTF = \frac{1}{Z(t)} \tag{5}$$

Failure prediction in software - example

In previous paragraph we discussed, theoretical mathematical concepts of failure predictions. The key figure in this equations is density function which describes distribution of failures in time. Earlier in previous paragraph, I also mentioned empirical distribution functions. In this paragraph, I will try to show how to calculate density function using empirical distribution function on real example from complex software system failures dataset.

Let's assume (and that is real example) that we have failures related data from complex software system (presented in the table below).

Table 1. Failures in system

Number of failures	1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	17	18	19	20	22
Time of failure	1	4	49	50	52	56	69	71	73	89	96	99	119	120	126	127	128	129	135	154

Number of failures	23	24	25	27	29	31	34	35	36	37	38	40	41	42	43	46	47	49	50	52
Time of failure	170	172	174	175	176	183	185	186	188	196	197	217	219	220	222	232	233	234	237	238

Number of failures	53	55	57	58	59	60	62	63	64	65	66	67	68	70	72	73	75	77	78	79
Time of failure	241	246	248	249	251	266	268	269	271	273	280	294	295	296	300	301	302	303	305	306

Number of failures	80	82	84	86	87	88	89	91	93	94	98	99	100	101	102	103	104	105	106	110
Time of failure	323	329	330	331	332	333	334	335	345	346	347	348	349	350	351	352	371	372	374	375

Number of failures	111	112	113	114	115	116	117	118	119	120	123	125	126	130	133	136	137	138
Time of failure	377	378	381	382	384	396	399	400	401	402	403	404	405	406	407	408	410	411

As can be seen, in system there had been observed 138 failures and the last was in 411 time interval. Data are form real system used by over 30 thousands of users every day, from January of 2005 to half of March of 2006. Therefore, data are from pretty modest system. Very often, we have that kind of data from our software systems.

At first we present some diagrams that show data from the table 1. First let's see how number of failures has increased in time.

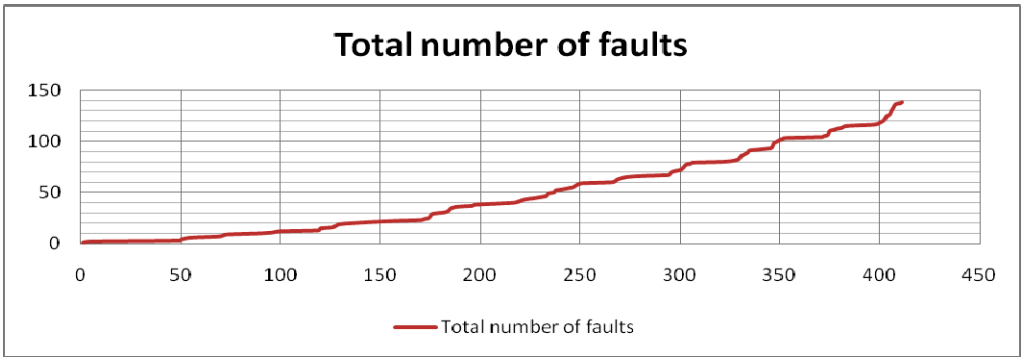

Figure 2. Total number of failures in time

Before we start computing distribution function, we can also look at faults intervals presented on figure 2.

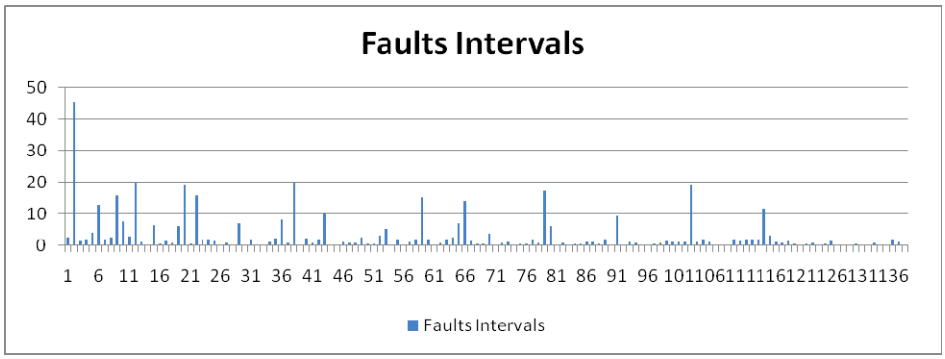

Figure 3. Faults intervals

At this moment we can start to do some calculations. Using data only from table 1, we can do some calculations which will describe quantitatively our examined software system. Calculated values are presented below.

- MTTF (average interval length)= 3 time intervals
- Median = 1
- Variance = 33,9
- Standard deviation = 5,8

As we see, on average we will have one fault every 3 days (we need to assume also, that time taking of failure is forgetful factor), but the variance of intervals is very big, so some of faults occurrences are dense and other time they are rare.

At this point, we can start with generating our density function using empirical distribution. First we need to group our fault interval into groups.

Table 2. Number of failures grouped by intervals

Interval	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**13**	**14**	**15**	**16**	**17**	**19**	**20**	**45**
Number of faults	39	48	20	7	2	1	3	3	1	1	1	1	1	1	1	2	1	2	2	1

If we will have more groups, then we would have more accurate approximation, but in our case 6 groups will be sufficient. Now we will group faults intervals into this 6 classes:

- $[0;1)$
- $[1;2)$
- $[2;3)$
- $[3;6)$
- $[6;10)$
- $[10;\infty)$

Table 3. Calculation of empirical distribution

Time interval	Number of faults	Percent of total %	Empirical distribution value
[0-1)	39	28,26%	0,28
[1-2)	48	34,78%	0,63

[2-3)	20	14,49%	0,78
[3-6)	10	7,25%	0,85
[6-10)	8	5,80%	0,91
[10-∞)	13	9,42%	1,00

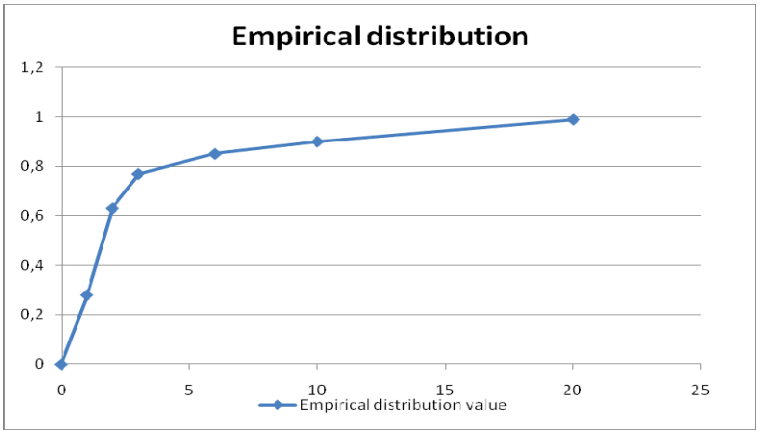

Figure 4. Empirical distribution value

Now when we have approximation of density function, we can easily go back to previous chapter and calculate probability of failures in the future. While calculating probability we need to read our values form above chart.

Bibliography

[Summerville] "Basic Reliability: An introduction to Reliability Engineering", Nicholas Summerville, AuthorHouse

[Wallace Coleman] „Application and Improvement of Software Reliability Models" , Dolores Wallace, Charles Coleman, SATC (Software Assurance Technology Center), NASA

[Grottke Dussa-Zieger] "Prediction of Software Failures Based on Systematic Testing", Michael Grottke, Klaudia Dussa-Zieger, University of Erlangen-Nuremberg

[NIST] http://www.itl.nist.gov, NIST - agency of the U.S. Commerce Department's Technology Administration.

[Podgurski Masri Yolanda] "Estimation of Software Reliability by Stratified Sampling", Andy Podgurski, Wassim Masri, Yolanda Mccleese, Francis G. Wolff, Charles Yang, Case Western Reserve University

[Crowe Feinberg] "Design for reliability", Dana Crowe, Alec Feinberg, CRC Press LLC

[Schneidewind] "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics", Norman F. Schneidewind, IEEE

[Everett] "Software Component Reliability Analysis", William W. Everett, SPARE INC.

[Lyu] "Software Reliability Theory", Michael Rung-Tsong Lyu, The Chinese University of Hong Kong

Author Information

Paweł Janczarek *– Institute of Computer Science, Faculty of Electronics and Information Technology, Warsaw University of Technology ; Nowowiejska 15/19, 00-665 Warszawa, Poland ; e-mail: pawel.janczarek@gmail.com*