RESEARCH INVENTY: International Journal of Engineering and Science
ISSN: 2278-4721, Vol. 1, Issue 3 (Sept 2012), PP 26-31
www.researchinventy.com

Fault Localization with Graphical Models in Software Engineering Applications

[1] V.Jyothsna, [2] R.Sailaja, [3] M.Koteshbabu
[1] *M.Tech STUDENT (SE), Sri sai aditya institute of Science & Technology, surampalem, JNTUK.*
[2] *Asst.Professor, Dept of CSE sri sai aditya institute of Science & Technologyg, surampalem, JNTUK.*
[3]*Software Engineer, Keane India Ltd., Hyderabad*

.

Abstract: *It costs more to solve and verify the problems in the product after the product was shipped. We know that a mistake leads to errors and errors leads us to faults. In this paper we explain the methods to solve the faults that are identified after products are shipped. We use a graphical method called probabilistic program dependence graph (PPDG), which facilitates probabilistic analysis and reasoning about uncertain program behavior, particularly that associated with faults, which derived from program dependence graphs (PDG).*

This Program Dependence Graphs (PDGs) had proven useful in software engineering applications such as testing, debugging and maintenance. This paper explains and uses PPDG, which is used to identify the faults and their probability of occurrence in product. This paper also indicates how PPDG is applied to identify and solve faults in both Team Software Process (TSP) and Personal Software Process (PSP) implementations and also in this paper we introduce different algorithms to identify the location where actual error had taken

Keywords- *Fault localization, Performance, Probabilistic Program Dependence Graph.*

I. Introduction

We have variety of graphical models, which have been used in software to identify the relation between the elements (statements) present in the program both structurally and dynamically. These models include flowcharts, control flow graphs, call graphs, finite state automata, data flow graphs and program dependence graph. By using these graphs we can find flow of information, errors during compilation (statistically) and during running (dynamically).Also by using PDGs we can know internal behavior of program.

These PDGs have been using in software application fields like testing, debugging and maintenance. Probabilistic graph models have proven successful in medicine, fraud detection, remote sensing and robotics due to their ability to model both the presence of certain dependence between variable of interest and the way in which the variables are probabilistically conditional on other variables. So hence we use these two concepts and build a new graph called Probabilistic Program Dependence Graph (PPDG).

Firstly, PPDGs are used and proven for fault localization and fault comprehension. In this paper we are using this concept for finding faults in software implementation models (i.e. TSP & PSP).By using this graph we can easily find where, why and by who faults were generated.

Later in this paper we explain how this PPDG can be used for identifying and solving faults in TSP and PSP models. The remainder of this paper is organized as follows: Section 2 presents background on models on which the PPDG is based, construction of PPDG and description of TSP and PSP. Section 3 presents the use of PPDG in TSP and PSP models. Section 4 illustrates the application by the two algorithms LocFault and IdenFault . Section 5 summarizes and section 6 discusses future work.

II. Background

Probabilistic Program Dependence Graph (PPDG) uses two models. First**, program dependence graph,** which represents structural dependences between program statements. Second, **dependence network**, which represents conditional dependence and independence relationships between random variables.

A program dependence graph (PDG) is a directed graph whose nodes represent program statements and whose edges represent data and control dependences. Labels on the control dependence edges represent the truth values of the branch conditions for those edges, and labels on data dependence edges represent the variables whose values flow along those edges.

A dependency network is a triple (S, G, Ω), where S represents a set of random variables, G = (N, E) is a possibly cyclic directed graph, and Ω represents a set of conditional probability distributions. N and E are the set of nodes and the set of directed edges in G, respectively, with nodes in G corresponding to random variables in S and edges in G representing dependences among the random variables.

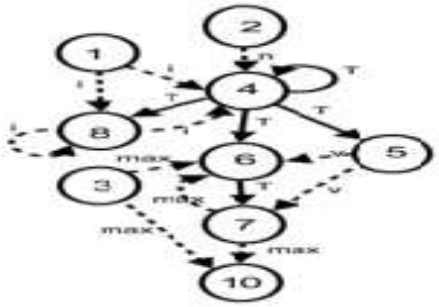

Fig 1: An example of program dependence

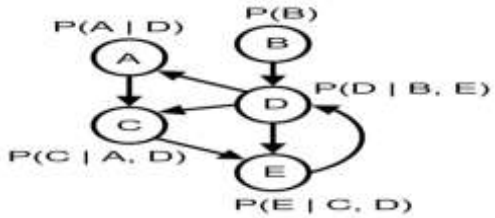

Fig 2: An example of dependency network

Fig 1 shows a sample example of PDG. Basically we convert the program statements in to nodes and these nodes are labeled with line numbers of corresponding statements in the program. Solid edges represent control dependences between nodes and dotted edges represent data dependences between nodes. For example, in fig 1, node 7 is control dependent up on node 6 and it is data dependent up on node 5. Fig 2 shows dependency network. For example a variable depends up on function output or a function takes input from another function output. Statements are taken in to nodes and flow of information (dependency) is denoted by arrows. It also shows conditional probability distribution for each node. For example, in fig 2, for node E, conditional probability distribution is P (E|C, D), that is node E is dependent up on nodes C, D.

2.1 Personal Software Process (PSP)

In certain ideal conditions, an individual specimen of a given software team would develop his own process model, so that, he can schedule all his activities, which will definitely be in favor of the organizations expectations, the same time model will take into account all the team members activities and also the organization on a whole.PSP make an individual to get involved in the project planning and also in maintaining the quality of all the software products on a whole. Five major framework activities of PSP are

- Planning, High level design, High level design review, Development and Postmortem.

2.2 Team Software Process (TSP)

Team software process aims in developing self directed project team, which is motivated effective software process, at the same time producing software which is of high quality. Generally maximum of 3-20 engineers with self motivated capable of analyzing their work form as team. Project leader or manager will be present for the entire team and he acts as leader. They work together to take team on the track of achieving CMM-5 targets. Important frame work activities required to be performed in TSP are

- Launch, High level design, Implementation, Integration, Test and Postmortem.

2.3 Construction of PPDG

A probabilistic program dependence graph (PPDG) is created by transforming the PDG of a program into a dependency network. The process of producing a PPDG consists of five main steps as shown in Fig 3.

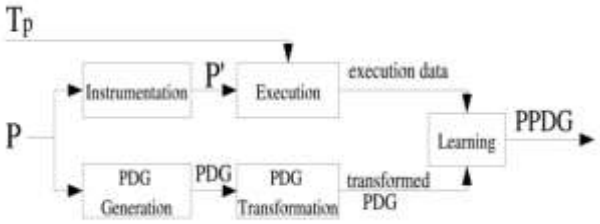

Fig 3: PPDG construction steps

First, the PDG-generation step generates the PDG of the input program P. Second, the PDG-transformation step takes the PDG, and transforms it by structurally changing the PDG and specifying states at nodes in the PDG, which results in a transformed PDG. Third, the Instrumentation step inserts probes into P to gather the execution data needed to estimate the parameters of the PPDG, and produces the instrumented program P0. Fourth, the Execution step executes P0 with its test suite T_p to generate the execution data. Finally, the Learning step generates a PPDG based on the execution data and the transformed PDG by estimating the parameters of the PPDG. The resulting PPDG is formally defined as follows, Probabilistic program dependence graph (PPDG) for program P is a triple (G, S, Q), where G=(N,E) is the transformed PDG of P whose node and edge sets are N and E, respectively, and S and Q are mappings from nodes to states and from nodes to conditional probability distributions, respectively.

III. Description

As explained in section2, in ***Personal Software Process*** model the implementation of Project/ Program is done individually by the programmers. The PSP Structure is shown in the Figure 4..The entire structure is presented in the form of hierarchy, as shown in diagram (basically we usually have more higher levels than shown in figure).The project manager is the person who takes responsibility to handle the project development and he acts as head and assign the units (modules) to the each and every member (Programmer).They take responsibility to implement the module by their own. In this way project development will be taken place in PSP model. Now programmers will develop the units which assigned to them by the project manager by their own with their knowledge levels. Here it takes lot of time to develop due to lack of communication between the programmers. They have to correct the errors and have to communicate with higher level people for suggestion. Figure 4 shows that Project Manager has number of programmers and Figure 5 shows its corresponding Dependence Graph. Here we have taken Project Manager as node X and we are assuming intermediate modules are there between the Project Manager and programmers and taking them as nodes A,B and C. Now we denote the programmers by taking them as nodes D, E, F, G, H, I, J, K and L. Now the dependency is calculated as shown in figure 5.

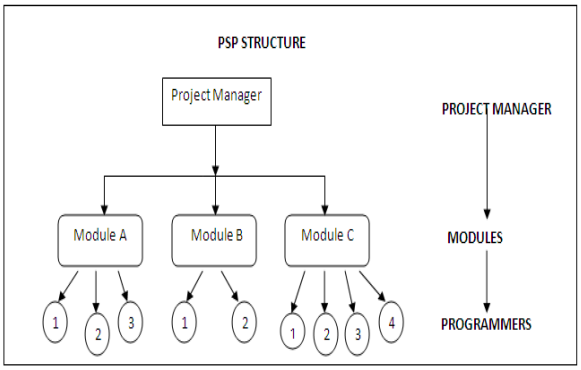

Fig 4: An example of PSP structure

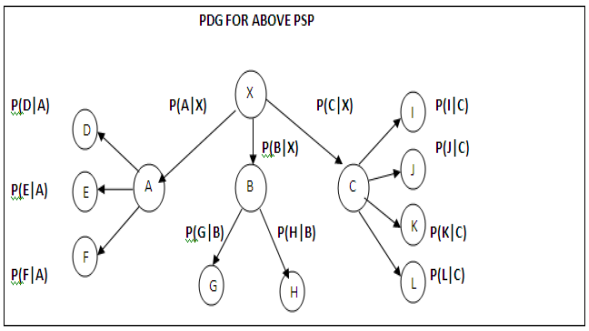

Fig 5: PDG for above PSP structure

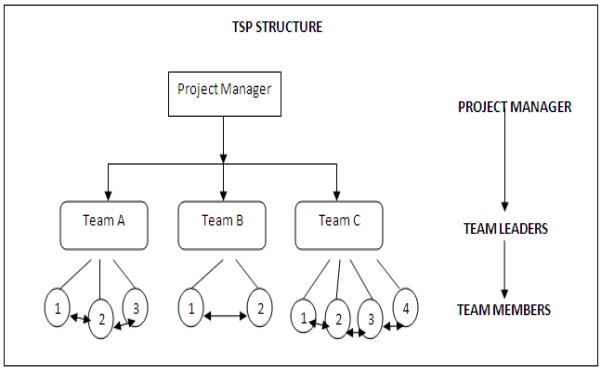

Fig 6: An example of TSP structure

For example node D is dependent on node A. So, dependency of node D is P (D|A).In this way we can calculate the dependency for all remaining child nodes.

Generally whatever may be techniques and precautions taken while developing the project errors will be crept in and after shipping they become faults and the organization has to be handling those faults again? Now, the organization has to do following things.

- Why fault has occurred?
- By whom it was entered?
- What is the amount of effect (Probability) due to that fault? Etc…

So, to find out above questions we have to gather the data and analyze that data. We state the table which shown below. As shown in table, in this format the data is collected and stored for analyzing.

S. I	Fault ID	Faults Explanation	Fault modules	Module developed by Team	Module developed by Team member
1	F324		M_a	A	A_1
2	F546		M_e	B	B_2
3	F121		M_b	A	A_2
4	F767		M_b	A	A_2
5	F237		M_g	C	C_2
6	F898		M_c	A	A_3
7	F318		M_b	A	A_2
8	F432		M_d	B	B_1
9	F376		M_c	A	A_3

As shown in table, for each fault a unique id is given and explanation about the fault is taken, fault modules are identified depending up on data collected. Then corresponding team, which developed that module (in case of TSP), corresponding programmer (team member) who developed that unit (module) in case of both TSP & PSP are collected.

As explained in section 2, ***Team Software Process*** model, the development or implementation is done by team coordination. The structure is shown in Figure 6. We had shown the communication between the team members by using bidirectional arrow(<-->).To generate its corresponding dependency Graph we make assumptions so that loops are not encountered and Now we can generate the dependency graph for above TSP same as above PSP.

IV. Application

In section 2, 3 we explained about the probabilistic program dependence graph (PPDG), PSP, and TSP. Now we are going to explain the actual application of finding faults in PSP, TSP models by using of PPDG.Here we propose two algorithms **LocFault** and **IdentFault.** First algorithm used to find the fault in the project and it is based on Bayesian model. LocFault algorithm is explained below In LocFault algorithm we give nodes as input and it gives fault nodes as output. It calculates the probability for all nodes and if lowest probability node is there then it is added to prob variable.

Algorithm: LocFault
Input: node-states:$\{X_j:x_{ji}\}^n_{j=1}$;PPDG
Output: fault nodes
  For all nodes 1 to n **do**
    find probability of each node
      If Prob < lowprob(Xj) **then**
        Lowprob(Xj) prob
        Assign node index num to node
        Configure the node(Xj)
      end
  end
  rank nodes by using probability
return ranked nodes with member ids.

Like this the above algorithm finds the faulty nodes in the models.

In Team Software Process model (TSP), the developers will communicate with each other and develop the modules. In this procedure the developer may implement the module with error and he may pass this error prone module to the next team. Now the next team may implement the module without notifying the error and pass it to next team and the process continues. Here we have to observe **two** cases. First the error may passed by the module developer itself and second it may passed from the other team. After the product shipped to customer, he notifies the fault and return back to the company. Now the organization stake holders have to find where the fault is located and by whom the fault is propagated and solution to it.

We provide the algorithm IdentFault, which discusses and identify the above two problems.

Algorithm : IdentFault
Input: ranked nodes with state configurations
Output : identifies who entered the error in the team
  For all ranked nodes 1 to n
    Watch prob calculated in LocFault algorithm
    Trace back the node with least prob up to starting of that fault **end**

V. Conclusion

We have discussed in this paper how to identify the faults in the TSP & PSP models. We illustrated the two algorithms, one LocFault which discusses the location of fault nodes in the model and second IdentFault, to identify the exact origin of faults done by the members of the model by considering the both cases (self generated error and error passed through from other team member).By using PPDG we can draw the graphs according to structures of the models and calculate the dependency easily. Hence PPDG is very useful in identifying the faults in software.

VI. Future Work

In future we will expand its usage in identifying the

- Performance of employers
- Accuracy of employers
- To find talent of employers of organization

And also we will expand PPDG applications like

- Giving rankings to organizations based on selected parameters
- Make it usable and available in different areas

References

[1] Y.Yu,J.A.Jones, and M.J.Harrod, "An Empirical Study of the Effects of Test-Suite Reduction on Fault Localization," Int'l conf. Software Engg, pp. 201-210, 2008.

[2] M.Renieris and S.Reiss, "Fault Localization with Nearest Neighbor Queries," Int'l conf. Automated Software Engg, pp. 30-39, Nov.2003

[3] A.Podgurski and L.A.Clarke, "A Formal Model of Program Dependences and Its Implications for Software Testing,Debugging, and Maintenance," IEEE Trans.Software Eng.,vol 16,no.9,pp.965-979,Sept.1990.

[4] J.Ferrante, K.J.Ottenstein, and J.D.Warren, "The Program Dependence Graph and Its Use in Optimization." ACM Trans Programming Languages and Systems, vol.9,no.3, pp.319-349, July 1987.

[5] George K. Baah , Andy Podgurski, and Mary Jean Harrold, "The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis," IEEE Trans.Software Eng., vol.36,no.4,July/August 2010.

[6] S.Bates and S.Horwitz, "Incremental Program Testing Using Program Dependence Graphs," Proc. Symp. Principles of Programming Languages,pp 384-396, Jan. 1993.