Proceedings of DETC'03:
ASME 2003 Design Engineering Technical Conference
and Computers and Information in Engineering Conference
Chicago, IL, September 2-6, 2003

DETC2003/DFM-48145

MODELING TEST, DIAGNOSIS, AND REWORK OPERATIONS AND OPTIMIZING THEIR LOCATION IN GENERAL MANUFACTURING PROCESSES

Zhen Shi
CALCE Electronic Products and Systems Center
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742

Peter Sandborn
CALCE Electronic Products and Systems Center
Department of Mechanical Engineering
University of Maryland
College Park, MD 20742

ABSTRACT

This paper presents a test, diagnosis, and rework analysis model for use in manufacturing process modeling. The approach includes a model of functional test operations characterized by fault coverage, false positives, and defects introduced in test, in addition to rework and diagnosis (diagnostic test) operations that have variable success rates and their own defect introduction mechanisms. The model accommodates multiple rework attempts on a product instance.

The model is applied within a framework for optimizing the location(s) and characteristics (fault coverage/test cost, rework success rate/rework cost) of Test/Diagnosis/Rework (TDR) operations in a general manufacturing process. A new search algorithm called Waiting Sequence Search (WSS) is applied to traverse a general process flow to perform the cumulative calculation of a yielded cost objective function. Real-Coded Genetic Algorithms (RCGAs) are used to perform a multi-objective optimization that minimizes yielded cost. An example of a general complex process flow is used to demonstrate the feasibility of the algorithm.

Keywords – Test economics; Test; Diagnosis; Rework; Cost modeling; Electronic systems.

INTRODUCTION

At a fundamental level, system design is a tradeoff analysis activity. This tradeoff includes factors such as size and performance, but often the most important factor in the tradeoff is cost. The various recurring costs that affect the manufacture of a system are the fabrication/assembly cost, test/inspection[1] cost, rework cost, and waste disposition cost.

For many types of systems, functional test is an important driver that significantly affects the total cost of manufacturing. In electronic systems, for example, it is not uncommon that greater than 60% of a product's recurring cost can be attributed to testing (Turino 1990), for integrated circuits, recurring testing costs are approaching 50% of the total product cost (Rhines 2002). When the products that result from a manufacturing process are imperfect, four costs are potentially involved, first the cost of determining whether a given instance of the product is good or bad (recurring functional testing), second the cost of determining what defect caused the faulty product and where the defect is located (diagnosis), third fixing the defect (rework), and fourth eliminating the causes of the defects (continuous improvement). Depending on the maturity of the product, its placement in the market, and the profit associated with selling it, all, some or none of the four activities listed above may be involved. Understanding the test/diagnosis/rework costs may determine the extent to which the system designer can control and optimize the manufacturing cost, and the extent to which it makes sense to do so.

The ultimate goal of any functional test strategy is the determination of: 1) When should a system be tested? At what point(s) in the manufacturing process? 2) How much testing should be done, i.e., how thorough should the test be? A test

[1] In this paper we are concerned with recurring functional (pass/fail) and diagnostic testing only, not environmental testing (i.e., qualification testing).

that detects 10% of the defects in a product may cost a small fraction of a test that identifies 95% of the defects, so, if I have multiple tests in a process, what is the optimum fault coverage to buy for each one? and 3) How much time and money should be spent to make the product more testable? These goals would be easy to realize if we had unlimited time, resources, and money. We could stop after every step in the manufacturing process and perform a full function test, and add structures to our system such that every critical element could be accessed and tested. These measures are unfortunately far from practical and we are usually faced with determining how to obtain the best test coverage possible for the least cost.

The specific goal of test economics is to minimize the cost of discarding good product and the cost of shipping bad product. This goal is enabled through the development of models that allow the yield and cost of products that pass through test operations to be predicted as a function of the properties of the product entering the test and the characteristics of the test operation (its cost, yield, and ability to detect faults in the product it is testing).

Although the model discussed here has general applicability to products fabricated using series and parallel processes, a specific application that can benefit immediately is electronic board assembly. Board assembly is process of attaching electrical components (chips, passives, connectors, etc.) to a printed circuit board to form a functional module (we will refer to an individual instance of product in the process as a "module" throughout the remainder of this paper).

There are several existing test/rework models that are applicable to process modeling and process-flow based cost analyses for electronic systems. Basic models, (e.g., Dislis *et al.* 1993, Tegethoff and Chen 1994, Scheffler *et al.* 1998) account for test fault coverage and single rework attempts (diagnosis is combined with rework and not explicitly treated). These models also assume that the application of the test does not contribute defects to the product, and that the test produces no false positives. More detailed models have also appeared. A model that appears in Abadir *et al.* 1994 and Sandborn and Moreno 1994, accomodates multiple rework attempts, but, again ignores defects introduced in testing, false postives, and although seperating diagnosis from rework diagramatically, does not actually treat diagnosis explicitly in its analysis. Athough the detail accomodated in the existing models varies, in general they do not account for new defects introduced during the test, treat diagnosis explicitly or false positives in testing.

In order to accommodate the additional effects and obtain a model that is readily useable in a process and cost modeling environment, a more comprehensive test/diagnosis/rework model has been formulated. The next section of this paper describes the model developed and used by the authors. The third section implements the model within a multi-objective optimization environment, and the forth section provides example results for optimizing the location and characteristics of test/diagnosis/rework operations in a general process flow.

TEST/DIAGNOSIS/REWORK MODEL

The objective of the test economics model is to accommodate the test/diagnosis/rework effects relevant to electronic system assembly processes. In these processes, defect insertion during test and rework operations is not uncommon (e.g., from handling and/or probes making physical contact with the board), false positives[2] can be a significant problem, multiple rework attempts are made when dealing with expensive systems such as multichip modules, and complex rework operations that may include reassembly of significant portions of the system are performed.

Figure 1 shows the content of the test/diagnosis/rework model. In the following description we use "module" to refer to the item being tested (e.g., a printed circuit board with chips assembled on it). Inputs to this model are the accumulated cost and yield of upstream processes (C_{in} and Y_{in}), the number of modules (N_{in}) is not a required input and is only included for convenience in the formulation of the model.[3] Yield is the ratio of non-defective (good) modules to all modules, non-defective and defective (bad). The test portion of the model is the top most group of three steps in Figure 1. This model can be used to account for defects introduced by the test operation both prior to the actual test (e.g., loading the module into the tester or stationing the probes on the module) and after the test result is recorded (e.g., unloading the module from the tester). The modules that are determined to be faulty go on to the diagnosis step. Three outcomes are possible from diagnosis: 1) no fault is found in which case the module goes back for retesting, 2) the module is determined to be reworkable and sent on to rework, or 3) the module is determined to be non-reworkable and sent to scrap. The rework process operates on the reworkable modules and scraps modules that can not be successfully reworked. The reworked modules are re-tested and if the reworked modules are found to be faulty again, the modules are again sent for diagnosis. This rework process can be performed a specified number of times (attempts).

There are several key assumptions made in the formulation of this model: defects introduced by the diagnosis step are not explicitly treated; and false positive (f_p) and fault coverage (f_c) act simultaneously and they are independent of each other, i.e., the fault coverage acts only on bad modules and the false positive acts either only on good modules or on all modules.

[2] A false positive is a positive test result in subjects who do not possess the attribute for which the test is conducted. Electronic systems test engineers define false positives as a Type I tester error (Williams *et al.* 1992). In testing, this means that a test will identify good product as bad at some non-negligible rate. In fact, data at the board and system level has shown that as many as 46% of all failures are not actually failures, but false positives, Henderson *et al.* 1992.

[3] In general, yield and cost results from this model are independent of N_{in}, however, if equipment, tooling, or other non-recurring costs are included, the results become dependent on N_{in} and can be computed from accumulations of time that specific equipment is occupied or the quantity of tooling used to produce a specific quantity of modules, e.g., see Trichy *et al.* 2001.

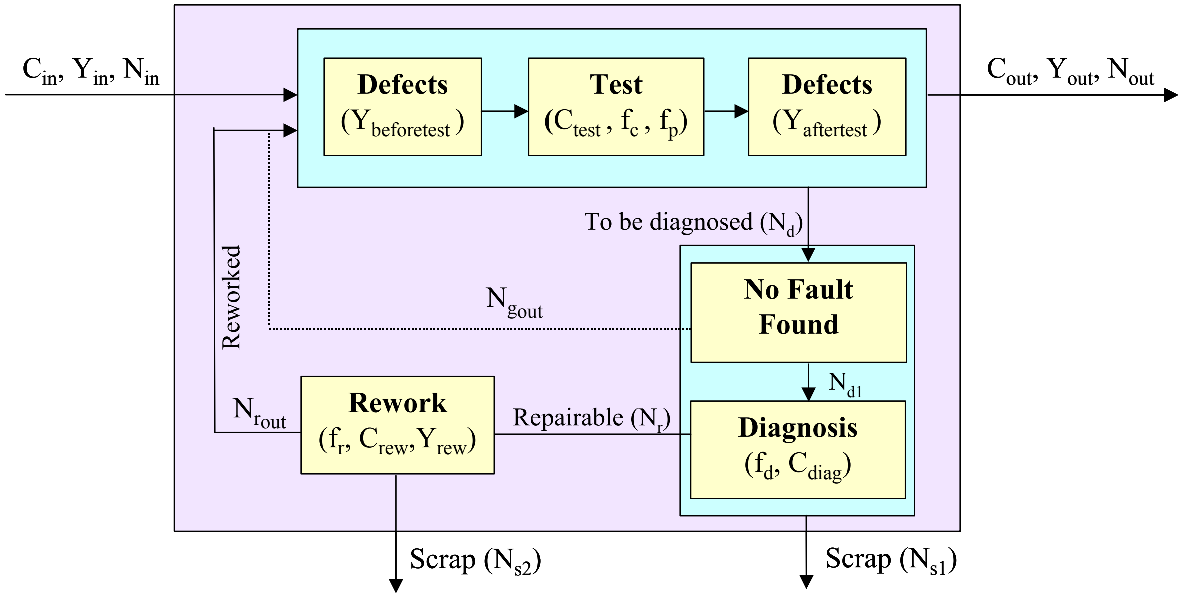

C_{in}, Y_{in}, N_{in}
Defects ($Y_{beforetest}$)
Test (C_{test}, f_c, f_p)
Defects ($Y_{aftertest}$)
C_{out}, Y_{out}, N_{out}
To be diagnosed (N_d)
No Fault Found
N_{gout}
Reworked
N_{d1}
Rework (f_r, C_{rew}, Y_{rew})
N_{rout}
Repairable (N_r)
Diagnosis (f_d, C_{diag})
Scrap (N_{s2})
Scrap (N_{s1})

Figure 1. Organization of the test/diagnosis/rework model. Table I describes the notation appearing in this figure.

Table I. Nomenclature used in Figure 1 and throughout the discussion in this section

C_{in}	Cost of a module entering the test/diagnosis/rework process	N_{in}	Number of modules entering the test/diagnosis/rework process
C_{test}	Cost of test/module	N_d	Total number of modules to be diagnosed
C_{diag}	Cost of diagnosis/module	N_{gout}	Number of no fault found modules
C_{rew}	Cost of rework/module	N_{d1}	$N_d - N_{gout}$
C_{out}	Effective cost of a module exiting the test/diagnosis/rework process	N_r	Number of modules to be reworked
f_c	Fault coverage	N_{rout}	Number of modules actually reworked
f_p	False positives fraction, the probability of testing a good module as bad	N_{s1}	Number of modules scrapped by diagnosis process
f_d	Fraction of modules determined to be reworkable	N_{s2}	Number of modules scrapped during rework
f_r	Fraction of modules actually reworked	N_{out}	Number of a modules exiting the test/diagnosis/rework process, includes good modules and test escapes
Y_{in}	Yield of a module entering the test/diagnosis/rework process		
$Y_{beforetest}$	Yield of processes that occur entering the test		
$Y_{aftertest}$	Yield of processes that occur exiting the test		
Y_{rew}	Yield of the rework process		
Y_{out}	Effective yield of a module exiting the test/diagnosis/rework process		

Versions of C_{in}, Y_{in} and N_{in} appear both with and without subscripts in the proceeding discussion. When the variables appear without subscripts they refer to the values entering the process. When they have subscripts, they represent specific rework attempts.

A. Cost Calculation

The cost incurred by all the modules that eventually pass the test step is given by,

$$C_1 = \sum_{i=0}^{n} \left(C_{in_i} + C_{test}\right) N_{out_i}, \tag{1}$$

where n is the number of rework attempts allowed, i.e., the maximum number of attempts to rework an individual module is n and N_{outi} is number of modules passed by the test in the ith rework attempt (see (6) and associated discussion). When i=0, C_1 is the total cost of the modules that pass the test without ever going through diagnosis or rework. The cost incurred by all the modules scrapped by the diagnosis step is given by,

$$C_2 = \sum_{i=1}^{n-1} \left(C_{in_i} + C_{test} + C_{diag}\right) N_{s1_i}, \tag{2}$$

and the cost incurred by all the modules scrapped by the rework step is given by,

$$C_3 = \sum_{i=1}^{n-1} \left(C_{in_i} + C_{test} + C_{diag} + C_{rew}\right) N_{s2_i}, \tag{3}$$

where N_{s1i} and N_{s2i} are defined in (8) and (9). After the final rework (nth rework attempt), the modules that do not pass the test are scrapped. The first term in (4) accounts for the defective modules scrapped by the final test, and the second term accounts for any false positives that are encountered during the final test,

$$C_4 = N_{d_n}\left(C_{in_n} + C_{test}\right) + N_{in_n} Y_{in_n} Y_{beforetest} f_p \left(C_{in_n} + C_{test}\right) \tag{4}$$

Equation (4) applies when f_p applies to only good modules, and when f_p applied to all modules. N_{inn} appearing in (4) is defined in (11). The total cost of all the modules (including scrapped modules) is the sum of C_1 through C_4. The total effective cost per output module associated with this model is the total cost divided by the total number of output modules (modules that are eventually passed by the test),

$$C_{out} = \frac{C_1 + C_2 + C_3 + C_4}{N_{out}}. \tag{5}$$

B. Quantity of Modules

The number of modules moving through different portions of the process is given by (6)-(11),

$$N_{out_i} = N_{in_i}\left(1 - f_p Y_{in_i} Y_{beforetest}\right)\left(\frac{\left(1 - f_p\right) Y_{in_i} Y_{beforetest}}{1 - f_p Y_{in_i} Y_{beforetest}}\right)^{f_c} \tag{6a}$$

$$N_{d1_i} = N_{in_i}\left(1 - f_p Y_{in_i} Y_{beforetest}\right) - N_{out_i} \tag{7a}$$

when f_p applies to only good modules, and

$$N_{out_i} = N_{in_i}\left(1 - f_p\right)\left(Y_{in_i} Y_{beforetest}\right)^{f_c} \tag{6b}$$

$$N_{d1_i} = N_{in_i}\left(1 - f_p\right) - N_{out_i} + f_p N_{in_i}\left(1 - Y_{in_i} Y_{beforetest}\right) \tag{7b}$$

when f_p applied to all modules.

$$N_{s1_i} = \left(1 - f_d\right) N_{d1_i} \tag{8}$$

$$N_{s2_i} = \left(1 - f_r\right) N_{r_i} \tag{9}$$

$$N_{r_i} = f_d N_{d1_i} \tag{10}$$

$$N_{in_i} = \begin{cases} N_{in} \text{ when } i = 0 \\ f_r N_{r_{i-1}} + f_p N_{in_{i-1}} Y_{in_{i-1}} Y_{beforetest} \text{ when } i > 0 \end{cases} \tag{11}$$

where parameters without subscripts (N_{in}, C_{in}, and Y_{in}) indicate values entering the process (Figure 1). Equation (6) follows from the classical result for the yield of modules that pass a simple test step (a test step that does not introduce new defects or false positives, and 0 rework attempts) given in terms of the incoming yield (Y_{in}) and the fault coverage (f_c), i.e., $Y_{out} = Y_{in}^{1-f_c}$, Willams and Brown 1981, and Agrawal *et al.* 1992. Using this result, it is can be shown that the fraction of modules starting the test that "pass" the simple test step is $Y_{in}^{f_c}$. The total number of modules that successfully pass the test process after n rework attempts is given by,

$$N_{out} = \sum_{i=0}^{n} N_{out_i}. \tag{12}$$

The module counting in (6)-(11) assumes that all false positives on good modules go through diagnosis and back into test without scrapping of modules in diagnosis or rework. The formulation is also only valid when $f_p < 1$, $Y_{in} > 0$ and $Y_{beforetest} > 0$. The input cost (C_{ini}) that appears in (1)-(4) is given by C_{in} when i = 0 and by (13) when i > 0.

$$C_{in_i} = \frac{\left(C_{in_{i-1}} + C_{test} + C_{diag}\right) f_p Y_{in_{i-1}} Y_{beforetest} N_{in_{i-1}}}{N_{in_i}} + \frac{\left(C_{in_{i-1}} + C_{test} + C_{diag} + C_{rew}\right) f_r N_{r_{i-1}}}{N_{in_i}}. \tag{13}$$

C. Yields

The input yield (Y_{ini}) that appears in (4) and (6)-(13) is given by Y_{in} when i = 0 and by (14) when i > 0.

$$Y_{in_i} = \frac{f_p Y_{in_{i-1}} Y_{beforetest} N_{in_{i-1}} + Y_{rew} f_r N_{r_{i-1}}}{N_{in_i}}. \tag{14}$$

The final yield of modules that successfully pass the process is given using the general result in Willams and Brown 1981, and Agrawal *et al.* 1992, by,

$$Y_{out} = \frac{\sum_{i=0}^{n} Y_{aftertest} N_{out_i} \left(\frac{\left(1 - f_p\right) Y_{in_i} Y_{beforetest}}{1 - f_p Y_{in_i} Y_{beforetest}}\right)^{1-f_c}}{N_{out}} \tag{15a}$$

when f_p applies to only good modules, and

$$Y_{out} = \frac{\sum_{i=0}^{n} Y_{aftertest} N_{out_i} \left(Y_{in_i} Y_{beforetest}\right)^{1-f_c}}{N_{out}} \quad (15b)$$

when f_p applied to all modules.

Note, N_{in} cancels out of (5) and (15) making the total cost per module and final yield independent of the number of modules that start the process, which is intuitively correct since no volume-sensitive effects (such as material or equipment costs) are included in the recurring model.

Examples of applying the model discussed in this section appear in Trichy *et al.* 2001.

D. Variable Rework Cost and Yield Models

In general the cost (time) of performing test depends on the fault coverage desired, and similarly, the cost of rework is linked to the yield of modules that result from it.

A relationship between the cost of test (proportional to test time or number of tests performed) and fault coverage has been suggested by Goel 1980. Empirical data shows that the test process can be divided in two phases. A relatively small subset of T_I (number of tests in Phase I see Figure 2) of the total set of tests provides a fault coverage ranging from 65% to 85% for most combinational logic circuits (Goel 1980). For Phase II of the test generation, the number of additional tests required is approximately a linear function of the number of untested faults remaining at the end of Phase I. Unlike Phase I, in Phase II each generated test tends to detect fewer faults than the one before it and the average cost per detected fault increases. For the purpose of simplifying the relationship, an exponential function can be used to approximately simulate Phase I and a linear function in Phase Π on the assumption that Phase II would never reach full coverage (100% of fault coverage) in practical testing. Assuming that the test cost is proportional to the number of tests, a relationship between test cost and fault coverage can be derived,

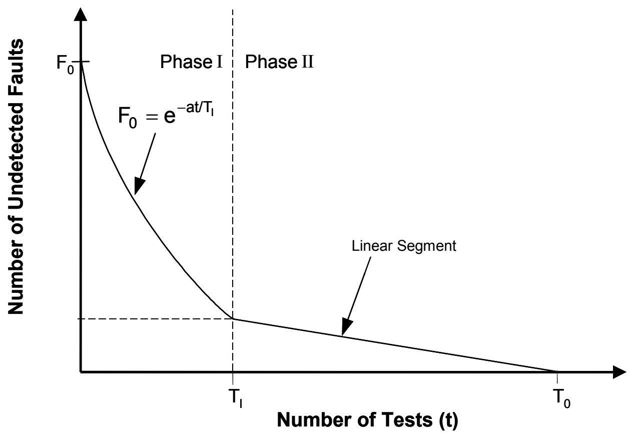


Figure 2. Typical curve of untested faults versus the number of tests, Goel 1980. "a" is a constant whose value is in the range 1-2 for a given logic structure.

$$C_{test} = p_t\left[b_t \ln(1-f_c) - r_t\right] + C_{ft},\ f_c \in (0,1) \quad (16)$$

where p_t is the cost coefficient; b_t is the coefficient of test characteristic, r_t is the fault ratio, f_c is the fault coverage of test, C_{ft} is the fixed cost of test[4].

A similar functional relationship between rework yield (y_r) and rework cost (C_{rew}) can be developed,

$$C_{rew} = p_r\left[b_r \ln(1-y_r) - r_r\right] + C_{fr},\ y_r \in (0,1) \quad (17)$$

The concept of having a choice of the fault coverage to purchase is straightforward – fault coverage is a measure of the ability of a set of tests (a collection of test vectors) to detect a given class of faults that may occur in a device under test, the fault coverage attained with a test is dependent on the number of test vectors exercised, which determines the test time and thereby the test cost. In the case of rework, the rework yield (really the rework success rate) depends on types of faults that are selected for repair and potentially the thoroughness of that repair.

Functional relationships between fault coverage and test cost, and rework yield and rework cost obviously depend on the type of system being considered, the particular baseline values used for the examples considered in this paper are shown in Table II. The relationships in (16) and (17) were used for the remaining work in this paper as <u>examples</u> only. The methodology that is the subject of this paper, will work successfully with alternative models.

Table II. Example values of factors in (16) and (17)

p_t	b_t	r_t	C_{ft}	p_r	b_r	r_r	C_{fr}
0.02	-288	8.2	1	0.02	-300	10	1

OPTIMIZING TDR LOCATIONS AND CHARACTERISTICS IN A PROCESS FLOW

The previous section develops a model for a single TDR operation. The next issue is to cumulatively compute the objective function of feature parameters (fault coverage/test cost, rework success rate/rework cost) of a general process flow that includes multiple TDR operations. For the purpose of derivation of the objective function, search algorithms are needed to traverse the process flow with the computation performed according to the sequence of process steps.

The framework for optimizing the implementation of TDRs in a process flow is shown in Figure 3. In the methodology, the process flow for manufacture or assembly of the electronic system is first generated. Then TDR operations are then inserted into the process flow at all possible locations (i.e., after every process step) for use as an initial guess. If the TDR operations can be inserted according to specific experiences the optimization will be more efficient. Starting with the process flow and the initial guess of TDR operation

[4] C_{ft} accounts for fixed costs associated with testing, i.e., there is a minimum fixed cost for having even a very small fault coverage.

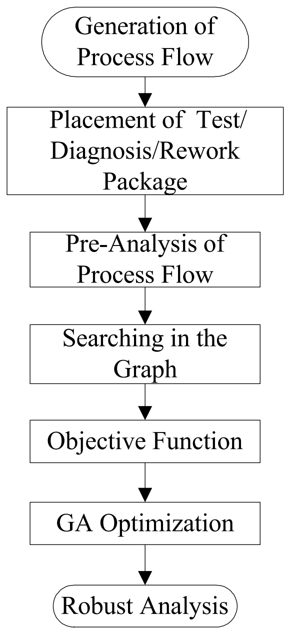

Generation of Process Flow
Placement of Test/ Diagnosis/Rework Package
Pre-Analysis of Process Flow
Searching in the Graph
Objective Function
GA Optimization
Robust Analysis

Figure 3. The framework for optimization of TDR operation location(s) and characteristics for a general process flow.

locations, a pre-analysis is executed to sort similar process steps and merge branches to decrease the complexity of process. A search algorithm is applied to traverse the entire process to perform the cumulative calculation of the objective function (yielded cost). Real-Coded Genetic Algorithms (RCGAs) are then used to perform a multi-objective optimization that minimizes yielded cost.

A. Graphical Representation of a Process Flow

Based on the analysis of the TDR model in the last section, we know how to compute the feature parameters of a single test step. A general process flow may, however, have many different (possibly independent) TDR activities located within it. The next issue to be addressed is how to obtain the objective function (yielded cost, i.e., cost divided by yield) for an entire general process flow. Graphs are useful in representing the process flow, i.e., complex systems involving binary relationships among process steps (Rao 1996).

A graph G consists of a set of nodes V and a set of edges E, $G = < V, E >$. Here G denotes the entire process flow, V denotes the process steps and an edge $< X, Y > \in E$ denotes the directed flow between two adjacent process steps. The degree of a node is the number of the neighbors adjacent to it. We write $X \to Y$ when $< X, Y > \in E$ is in a directed graph (DIGRAPH) (Rao 1996, Choi and Chatterjee 2001). We define $\mathrm{PRED}(X)$ as the set of all predecessors (process steps preceding X) of node X, and $\mathrm{SUCC}(X)$ as the set of successors (process steps after X) of X. if $X \to Y$, then $X \in \mathrm{PRED}(Y)$

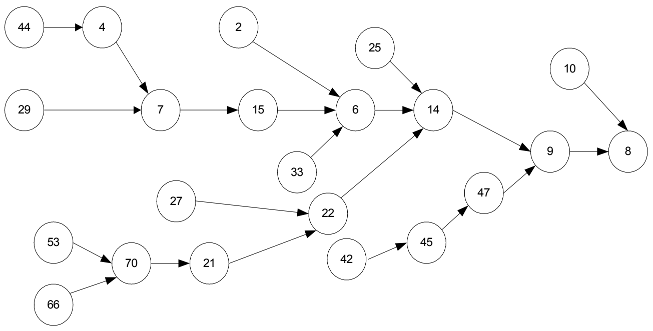


Figure 4. Example graphical representation of a complex process flow.

and $Y \in \mathrm{SUCC}(X)$. The indegree $\mathrm{id}(v)$ of a node X is the cardinality of $\mathrm{PRED}(X)$ and the outdegree $\mathrm{od}(v)$ of a node X is the cardinality of $\mathrm{SUCC}(X)$. The graphical representation of an example complex process flow is shown in Figure 4.

From the analysis of a generic process flow like the one in Figure 4, four types of basic steps have been identified:

- Start Step, $\mathrm{od}(v) = 1$ and $\mathrm{id}(v) = 0$, There are no inputs to the Start Step and just one output from it. There may be multiple Start Steps in a complex process flow.
- Sequence Step, $\mathrm{od}(v) = 1$ and $\mathrm{id}(v) = 1$, There is one input and one output for a sequence step. Sequence Steps are the most common type of step in process flows for electronic systems.
- Cross Step, $\mathrm{od}(v) = 1$ and $\mathrm{id}(v) \geq 1$, There are multiple inputs and just one output associated with this kind of step. The complexity of the problem is significantly increased by each Cross Step in the process flow.
- End Step, $\mathrm{od}(v) = 0$ and $\mathrm{id}(v) \geq 1$, An End Step represents the end of the process flow, which merges all the branches to one. The objective function of the process flow is derived from an End Step. There may only be one End Step in a process flow.

B. Waiting Sequential Search (WSS)

To derive the objective function of a process flow, every process step needs to be searched in order to perform the cumulative computation. Significant previous work on graph-based search algorithms exists, e.g., Chartrand and Oellermann 1993, Tarjan 1972. Several algorithms have been applied to resolve search problems similar to the one posed here, Rao 1996, Choi and Chatterjee 2001. As to the graph-based representation of our process flows, there are several specific features that make it attractive to propose a new search algorithm to perform an efficient search in the process flow.

From the Figure 4, the following features of the general complex process flow can be observed: 1) The $\mathrm{od}(v) \leq 1$ is always true for all the vertices (process steps) in the process flow; and 2) There are sequential search requests for the graph, i.e., only when all the predecessors PRED(Y) of v(Y) have been visited, then v(Y) could be visited.

WSS begins from the lowest-numbered vertex that belongs to a Start Step then proceeds to search the next step. By checking the type of the successor, the algorithm decides to continue to search to the next Sequence Step or wait at the Cross or End steps. After moving to the next step, the corresponding computation of feature parameters of the previous step is performed and the outcome is stored in a data table in which all the property information associated with process steps are recorded. If Cross or End types of steps are encountered, the visitation status of all predecessors will be checked. If all the predecessors have been checked, the searching continues, if not, the search begins from another Start Step type of vertex until the last Start Step vertex is visited. The algorithm requires that the searching of the next step continue only after all the branches of the present step have been visited. There are waiting actions for the sequential search based on the characteristics of the process flow. The searching process for the complex process flow example shown in Figure 4 using WSS is described below:

1. First, begin from the lowest number of Start Steps (2), (2) → (6), check whether all the other branches have been searched, compute then wait;
2. (10) → (8), check the other branches and wait;
3. (25) → (14), wait;
4. (27) → (22), wait;
5. (29) → (7), wait;
6. (33) → (6), wait;
7. (42) → (45) → (47) → (9), wait;
8. (44) → (4) → (7) → (15) → (6) → (14), wait;
9. (53) → (70), wait;
10. (66) → (70) → (21) → (22) → (14) → (9) → (8), (8) is an End Step, all the branches have met and the process ends.

C. Multi-Objective Optimization Function

The objective of optimizing TDR location(s) and characteristics to be minimized is the yielded cost of the entire process flow (Becker and Sandborn 2001). The yielded cost we are interested in is the final effective cost per product instance (after the final processing step and/or TDR operation) divided by the final product yield. This yielded cost gives a measure of the effective cost per good product instance after all the manufacturing and TDR operations are completed. The final cost per product instance and yield are determined by accumulating (sum or product) the individual process step costs and yields and the TDR operation costs and yields (equation (5) and (15)) in the appropriate sequence through the process.

The objective function in which feature parameters of all possible TDR operations are considered can be derived from sequential cumulative computation from the Start Steps to the End Step. When the WSS algorithm traverses the entire process flow, a cumulative function is computed to be used as the objective function that needs to be minimized in the optimization. The optimization problem becomes,

$$\min_{x \in X} \sum_{i=1}^{n} C_Y(x_{1_i}, x_{2_i}, \cdots x_{m_i}) \tag{18}$$

where,

m = number of feature parameters to be optimized;
n = number of total process steps with all possible TDR operations included;
C_Y = yielded cost of the process flow, cumulative cost divided by final yield.

In the optimization, first the TDR operations are placed in all possible locations or be chosen according to expert suggestions. The fault coverage (f_{ci}), rework yield (Y_{ri}) and rework attempts for the ith TDR operation need to be optimized in order to minimize the total yielded cost of the process flow. For example, for the complex process flow in Figure 4, there are 22 (including the one location after the End Step - 8) possible TDR operation locations following each of the process steps if there were no specific location constraints as to where a TDR operation could or could not be placed provided by the user. If just fault converge and rework yield of the TDR operations are to be optimized, the objective function can be written as (19),

$$\min_{f_{c_i}, Y_{r_i} \in X} \sum_{i=1}^{22} C_Y(f_{c_i}, Y_{r_i}),\ X \in [0,1]. \tag{19}$$

D. Optimization with Real-Coded Genetic Algorithms (RCGAs)

Complex process flows with hundreds of process steps are not uncommon. A general optimization of such a process flow requires an equally large stream of TDR operations resulting in several hundred variables to be optimized for the tradeoff analysis. To optimize feature parameters of possible TDR operations in order to find the global optimum of yielded cost in the process flow, RCGAs are used. RCGAs have the advantage of requiring less storage than the Binary-Coded Genetic Algorithms (BCGAs) and the accurate representation of continuous parameters, which enables efficient optimization of multi-parameter functions (Oyama *et al.* 1999).

With RCGAs integrated, a process flow cost optimization modeling system according to the framework in Figure 3 has been developed to determine the optimum placement and characteristics of TDR operations in general process flows. The system results in suggestions of where the TDR operations should be inserted and not inserted, and what the optimal values of feature parameters of testing and rework should be based on the optimization analysis of the objective function.

OPTIMIZATION IN A COMPLEX PROCESS FLOW

For demonstration, the process flow shown in Figure 5 has been used. All possible TDR locations have been marked in the process. Table III shows the corresponding process step properties (cost and yield); the characteristics of the TDR operations 14-24 are solved for in the optimization process. The cost and yield associated with the individual process steps is

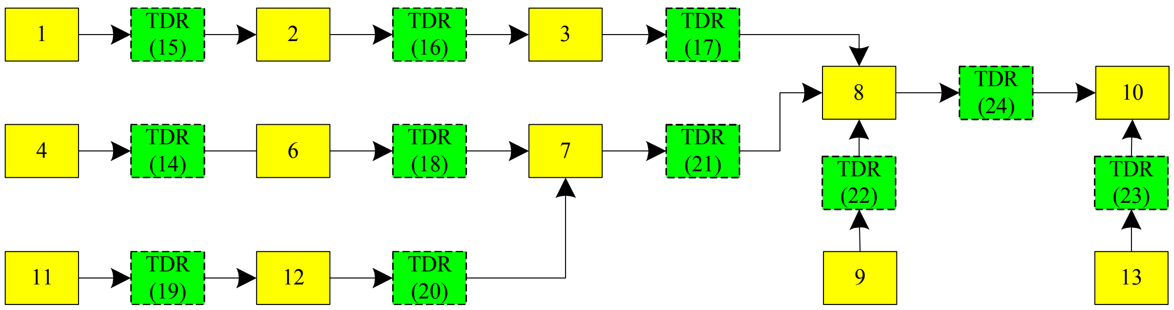


Figure 5. An example process flow with all possible TDR operations.

Table III. Characteristics of the process steps in Figure 5 (note, there is no step 5)

Process Step	Cost ($)	Yield (fraction)
Input to 1	41	0.91
Input to 4	60	0.36
Input to 9	37	0.42
Input to 11	9	0.95
Input to 13	75	0.96
1	21.1	0.95
2	12.3	0.88
3	14	0.89
4	23.8	0.94
6	45	0.96
7	11.3	0.86
8	33.8	0.92
9	13	0.9
10	15	0.92
11	60	0.95
12	78	0.91
13	54	0.94

example data that is representative of high-end electronic system assembly. As an example comparison analysis, the optimum fault coverage and rework yield of all possible TDR operations for various fixed cost values of test and rework are shown in Figure 6. The test and rework step properties are given in Table II except for the values are C_{ft} and C_{fr}, which are varied as indicated in Figure 6.

The algorithm begins by automatically placing TDR operations in all possible locations: 14-24 in Figure 5. The algorithm then determines the fault coverage and rework characteristics of each TDR that minimize the final yielded cost. Figure 6 shows that results from the optimization for different assumptions about the fixed costs associated with the test and rework. Figure 6(a) has low (inexpensive) test and rework - C_{ft} and C_{fr} are both small; as a result, 9 of the 11 possible locations for tests are present (i.e., have fault coverages above the threshold for testing, which is a fault coverage of 0.1). Because rework is also inexpensive in case (a), rework is being done at all the actual test locations. The Figure 6(a) result is intuitive, if test and rework are inexpensive, then test and rework will be done after nearly every process step. Figure 6(b) has inexpensive testing (same as case (a)), but the rework is expensive. As a result, significantly fewer rework opportunities are actually exercised. Notice also that the optimum test locations (and fault coverages) change due to the inclusion or exclusion of rework possibilities, even though the characteristics of the testing are the same. Figure 6(c) shows the same test costs as 6(a) and (b), but no rework is allowed – the optimum test locations and fault coverages again differ from cases 6(a) and 6(b) and the average value of fault coverage increases to compensate for the loss of rework capabilities (the optimization algorithm is attempting to maximize the final yield in order to minimize cost divided by yield). Figure 6(d) has expensive test and expensive rework. As a result, only 3 of 11 possible test locations are used (Test 20, 23 and 24 only), however, rework is included with all three of these tests. In this case, the majority of the testing is focused on Test 24 near the end of the process indicating that if test is expensive and defects introduced in the process are spread over the entire process (as opposed to being focused on a single process step), the optimum test location is near the end of the process, which is intuitive.

Figure 6 provides example optimization results for an example process flow. Figure 7 shows example convergence characteristics of the yielded cost of the complex process flow with the number of generations of the RCGAs.

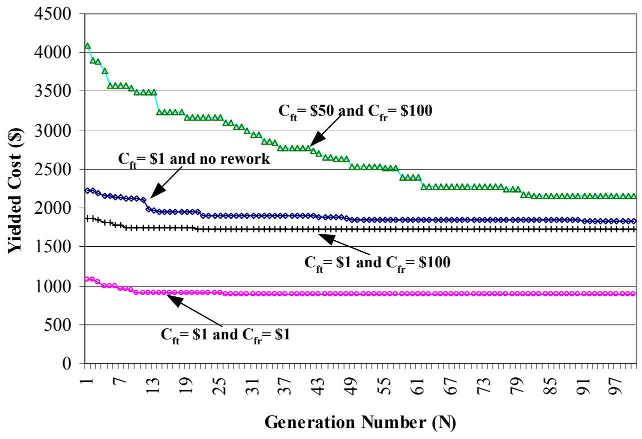
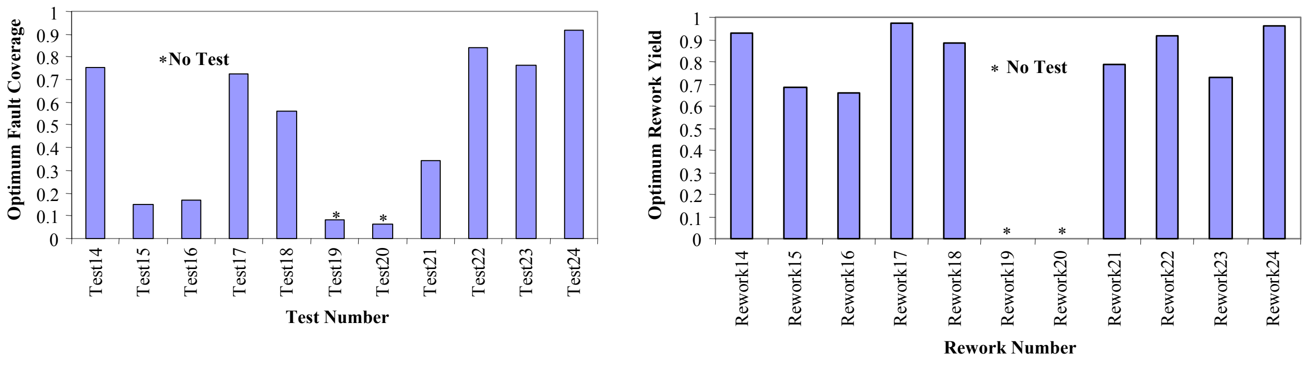
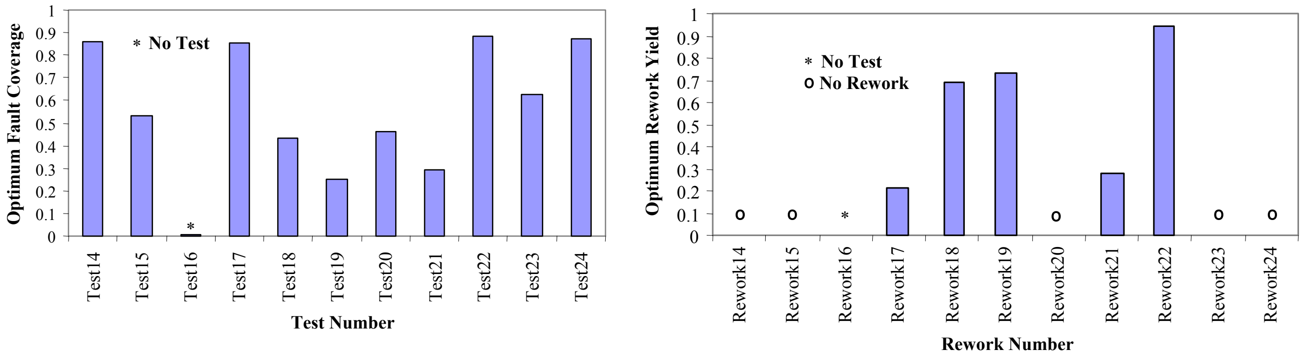


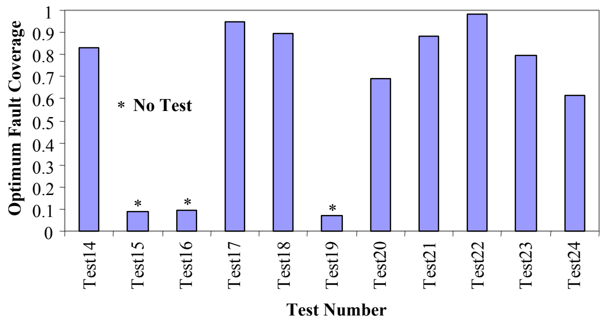
Figure 7. Optimization of yielded cost for various levels of fixed cost of test and rework



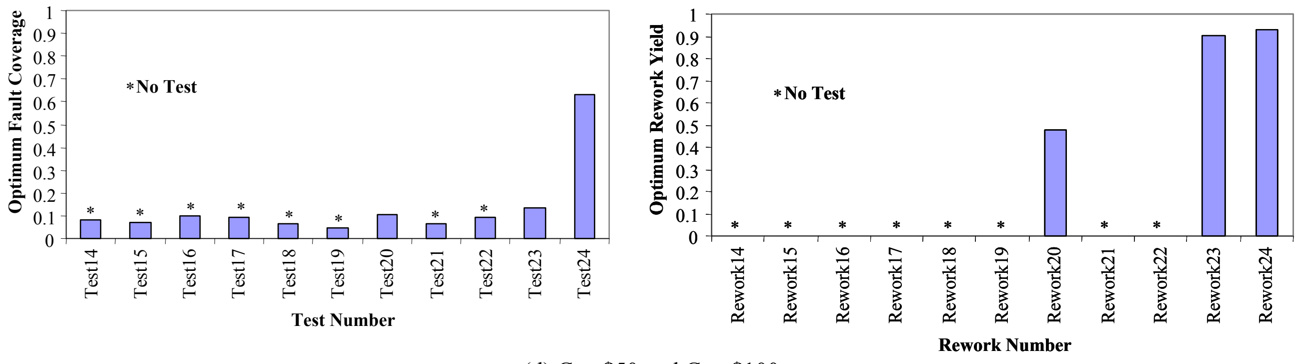
(a) C_{ft}= \$1 and C_{fr}= \$1



(b) C_{ft}= \$1 and C_{fr}= \$100



(c) C_{ft}= \$1 and no rework



(d) C_{ft}= \$50 and C_{fr}= \$100

Figure 6. Computed optimum feature parameter values for TDR operations in the process flow shown in Figure 5. Various fixed test and rework costs used in (16) and (17) assumed, other parameters given in Table II.

SUMMARY

This paper describes a framework for the optimization of Test/Diagnosis/Rework (TDR) location(s) and characteristics in a manufacturing processes A new search algorithm is developed and used to analyze complex process flows and to obtain values of a multi-objective function with a stream of feature parameters included. An optimization modeling system with Real-Coded Genetic Algorithms (RCGAs) integrated has been developed to optimize critical parameters and possible TDR locations for general process flows. The methodology developed and demonstrated in this paper guides the placement of TDR operations in practical manufacturing processes and has been specifically applied to electronic systems manufacturing (board assembly). The ability to optimize the TDR operations can also be used as the feedback to a Design for Test (DFT) analysis of the electronic systems showing which portion should be redesigned to accommodate the testing for a higher level of fault coverage and where there is less need for test to decrease the cost of products.

REFERENCES

Abadir, M., Parikh, A., Bal, L., Sandborn, P., and Murphy, C., "High Level Test Economics Advisor," *Journal of Electronic Testing: Theory and Applications*, 1994, Vol. 5, No. 2&3, pp. 195-206.

Agrawal, V., Seth, S., and Agrawal, P., "Fault Coverage Requirement in Production Testing of LSI Circuits," *IEEE J. of Solid-State Circuits*, 1982, Vol. SC-17, No. 1, pp. 57-61.

Becker, D., and Sandborn, P., "One the Use of Yielded cost in Modeling Electronic Assembly Processes," *IEEE Trans. on Electronics Packaging Manufacturing*, July 2001, Vol. 24, pp. 195-202.

Chartrand, G., and Oellermann, O.R., *Applied And Algorithmic Graph Theory,* McGraw-Hill, Inc., 1993.

Choi, K., and Chatterjee, A., "Efficient Instruction-level Optimization Methodology for Low-Power Embedded Systems," *Proc. of the 14th Int. Symposium on System Synthesis,* 2001, pp. 147-152.

Dislis, C., Dick, J.H., Dear, I.D., Azu, I.N., Ambler, A.P., "Economics Modeling for the Determination of Test Strategies for Complex VLSI Boards," *Proceedings of the International Test Conference*, 1993, pp. 210-217.

Goel, P., "Test Generation Costs Analysis and Projections," *Proc. of Design Automation Conference,* 1980, pp. 77-84.

Henderson, C.L., Williams, R.H., and Hawkins, C.F., "Economic Impact of Type I Test Errors at System and Board Levels," *Proceedings of the International Test Conference*, 1992, pp. 444-452.

Oyama, A., Obayashi, S. and Nakahashi, K., "Wing Design Using Real-coded Adaptive Range Genetic Algorithm," *Proceedings of 1999 IEEE Int. Conf. on Systems, Man, and Cybernetics,* 1999, Vol. 4, pp. 475-480.

Rao, N., "On Parallel Algorithms for Single-Fault Diagnosis in Fault Propagation Graph Systems," *IEEE Trans. on Parallel and Distributed Sys.,* Dec. 1996, Vol. 7, pp. 1217-1223.

Rhines, W., Keynote address at the Semico Summit, Phoenix, AZ, March 2002.

Sandborn, P.A., and Moreno, H., *Conceptual Design of Multi-chip Modules and Systems*, Kluwer Academic Publishers, Boston, 1994.

Scheffler, M., Ammann, D., Thiel, A., Habiger, C., and Troster, G., "Modeling and Optimizing the Costs of Electronic Systems," *IEEE Design & Test of Computers*, 1998, Vol. 15, No. 3, pp. 20-26.

Tarjan, R.E., "Depth-First Search and Linear Graph Algorithms," *SIAM J. Computing*, 1972, Vol. 1, No.2, pp.146-160.

Tegethoff, M., and Chen, T., "Defects, Fault Coverage, Yield and Cost, in Board Manufacturing," *Proceedings of the International Test Conference*, 1994, pp. 539-547.

Trichy, T., Sandborn, P., Raghavan, R., and Sahasrabudhe, S., "A New Test/Diagnosis/Rework Model for Use in Technical Cost Modeling of Electronic Systems Assembly," in *Proceedings of the International Test Conference*, Nov. 2001, pp. 1108-1117.

Turino, J., Design to Test – A Definitive Guide for Electronic Design, Manufacture, and Service, Van Nostrand Rienhold, New York, NY, 1990.

Williams, R.H., Wagner, R.G., and Hawkins, C.F., "Testing Errors: Data and Calculations in an IC Manufacturing Process," *Proceedings of the International Test Conference*, 1992, pp. 352-361.

Williams, T.W., and Brown, N.C., "Defect Level as a Function of Fault Coverage," *IEEE Transactions on Computers*, 1981, Vol. C-30, No. 12, pp. 987-988.