

Representing Troubleshooting Information  
for a High-Volume Production Line

by

Catherine M. Brennan

S.B., Computer Science and Engineering, MIT, 1984

Submitted to the Sloan School of Management and  
the Department of Electrical Engineering and Computer Science  
in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Electrical Engineering and Computer Science

and

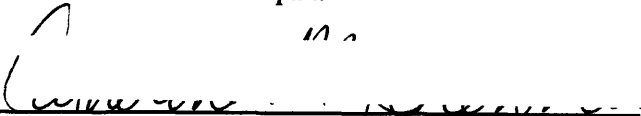
Master of Science in Management

at the  
Massachusetts Institute of Technology  
May, 1994

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part.

© Catherine M. Brennan, 1994 All rights reserved


Signature of Author

  
MIT Sloan School of Management  
May 7, 1994


Certified by

  
Alvin W. Drake, Professor of Systems Science and Engineering  
Thesis Supervisor

Certified by

  
Thomas W. Eagar, POSCO Professor of Materials Engineering  
Thesis Supervisor


Certified by

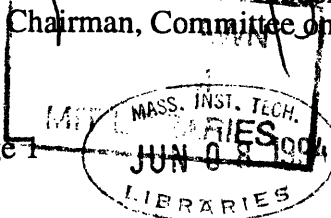
  
Donald Rosenfield, Senior Lecturer, Sloan School of Management  
Thesis Supervisor

Accepted by

  
Jeffrey A. Barks  
Associate Dean, Sloan Master's and Bachelor's Programs

Accepted by

  
Frederic R. Morgenthaler  
Chairman, Committee on Graduate Students





Representing Troubleshooting Information  
for a High-Volume Production Line

by

Catherine M. Brennan

Submitted to the Sloan School of Management  
on May 7, 1994  
in partial fulfillment of the requirements for the degree of  
Master of Science in Management

**ABSTRACT**

This thesis presents the concept of a troubleshooting tree, a semi-automated problem-solving tool that falls somewhere between a fishbone diagram and a decision tree. Troubleshooting trees are appropriate for problems which require a substantial amount of time to track down the root cause, and for which there are people who can identify the majority of the root causes. Furthermore, the environment in which the troubleshooting tree is to be used must be one where the people who encounter the problems understand the domain and are able to interact within it reasonably well. Various implementation approaches are considered, including AI representations of diagnostic problems as well as graphical representations of cause and effect information. The information needed to build a troubleshooting tree is identified, and a procedure is presented for collecting and organizing this information.

Thesis Supervisors: Alvin W. Drake, Professor of System Science and Engineering  
Thomas W. Eagar, POSCO Professor of Materials Engineering  
Donald Rosenfield, Senior Lecturer, Sloan School of Management



The author wishes to acknowledge the Leaders for Manufacturing Program for its support of this work.

Thanks is also due to the Hewlett-Packard Company for the use of their facilities and the open support of their people. Tony McGettigan, Ted Barnes, George Custer, and Gary Siewell, as well as all of the engineers developing the "M" production line are singled out for special thanks.

In addition, this thesis and the studies that led to it would not have been possible without the support of Digital Equipment Corporation and its Graduate Education in Manufacturing program.

Finally, the author expresses her gratitude for the support and guidance provided by her thesis advisors and other members of the MIT faculty who provided assistance in the completion of this thesis. Thank you to Al Drake, Norm Rasmussen, Don Rosenfield, and Tom Eagar.



<b>1. Introduction .....</b>	<b>9</b>
1.1. Contributions .....	9
1.2. Background .....	9
1.3. Definition of the problem .....	9
1.4. What is a troubleshooting tree? .....	9
1.5. Motivation .....	11
1.6. Overview .....	12
<b>2. Current Problem Solving Techniques .....</b>	<b>14</b>
2.1. Ink-jet pen production at HP Corvallis .....	14
2.1.1. The product: A thermal ink-jet pen .....	14
2.1.2. The manufacturing process .....	16
2.2. Common problems on the production lines .....	18
2.3. How production problems are noticed .....	20
2.4. How production problems are solved .....	21
2.4.1. People's roles in the manufacturing process .....	22
2.4.2. Current problem-solving tools .....	23
2.5. The benefits of providing troubleshooting tools .....	24
<b>3. Alternative Representations of the Information .....</b>	<b>26</b>
3.1. AI representations for diagnostic problem solving .....	27
3.1.1. Model-based reasoning .....	27
3.1.2. Case-based reasoning .....	29
3.1.3. Decision trees .....	30
3.1.4. Bayesian updating .....	31
3.2. Representations of cause and effect .....	31
3.2.1. Graphical .....	32
3.2.2. A database with a forms interface .....	33
3.2.3. Hypertext .....	34
3.3. The recommended representation .....	34
<b>4. Selecting a Subset of the Line for Which To Build a Troubleshooting Tree .....</b>	<b>37</b>
4.1. Possible "roots" for a troubleshooting tree .....	37
4.1.1. Failures at an inspection station .....	38
4.1.2. Defects at an audit station .....	39
4.1.3. Processes .....	40
4.1.4. Dataliner messages .....	41
4.2. Criteria for selection .....	41
4.3. Comparison of the alternatives .....	42
4.3.1. Failures at an inspection station .....	43
4.3.2. Defects at an audit station .....	46
4.3.3. Processes .....	47
4.3.4. Dataliner messages .....	47
4.4. The chosen problem .....	49
<b>5. Collecting Problem Solving Information .....</b>	<b>50</b>
5.1. The information required to create a troubleshooting tree .....	50
5.2. FMEAs as a source of troubleshooting information .....	51
5.2.1. Description of FMEA .....	52
5.2.2. Appropriateness of FMEA .....	53
5.3. A procedure for collecting troubleshooting information .....	56
<b>6. Results and Conclusions .....</b>	<b>63</b>
6.1. Follow-on work by others .....	63
6.2.1 Description of the tool .....	63
6.2.2 Current users .....	64
6.2.3 Software implementation difficulties .....	65
6.2.4 Management reactions .....	67
6.2 Cost/benefit analysis .....	67

6.3. Recommendations .....	70
6.4. Conclusions .....	73
<b>Bibliography .....</b>	<b>75</b>



# **1. Introduction**

## **1.1. Contributions**

This thesis presents the concept of a troubleshooting tree, a simple tool to aid in solving problems on a production line, which is nevertheless sufficiently powerful in certain cases. The data necessary to build a troubleshooting tree are identified, and a procedure for collecting the data is documented. The thesis includes data on the faults that could cause two production problems on a specific production line. For the more complex problem, 82 root causes are identified.

## **1.2. Background**

The research for this thesis was done at Hewlett-Packard's site in Corvallis, Oregon, where ink-jet pens for HP's Thermal Ink-Jet (TIJ) printers are manufactured. The work was done in conjunction with a Manufacturing Development group that is designing the tooling for a production line to make a new product, called the "M" pen.

## **1.3. Definition of the problem**

The problem addressed in this thesis deals with capturing information from the engineers who are designing a new line and using it to help people solve problems that occur on the line. What are the sources of this information? How can one gather the necessary information? What should be included? How should the information be represented?

## **1.4. What is a troubleshooting tree?**

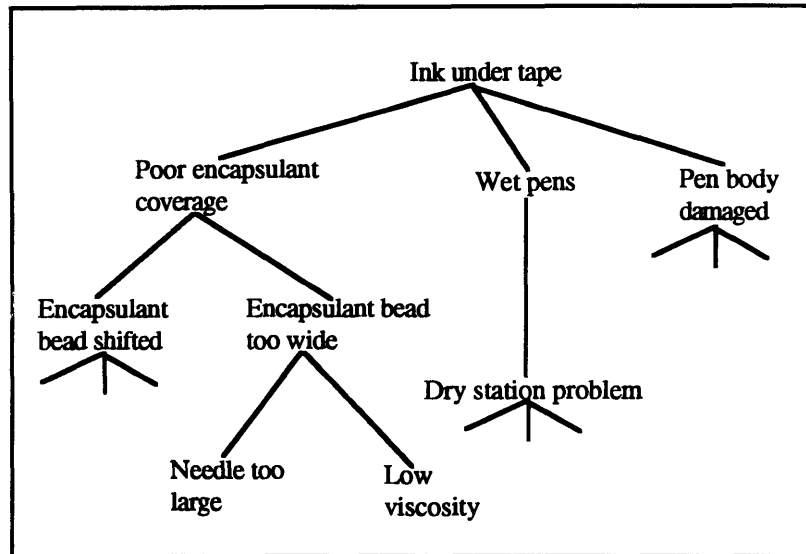
In this thesis the concept of a "troubleshooting tree" is used to describe the information that is needed to assist people in solving production problems. A troubleshooting tree is

a graph<sup>1</sup>, whose roots are problems, whose nodes are possible causes, and whose leaves are root causes.

Figure 1 presents a sample troubleshooting tree for the problem of "ink under tape." There are three broad categories of problems that could possibly have led to there being ink under the tape: poor encapsulant coverage; wet pens; or a damaged pen body. Each of these problems is further expanded into its possible causes<sup>1</sup>, and so on. The only root causes illustrated in this example have to do with the encapsulant: the needle might have been too large; or the encapsulant of too low a viscosity. Associated with these two root causes would be solutions. If the actual cause of ink under tape were found to be too large a needle dispensing the encapsulant, switching to the appropriate sized needle might be the recommended solution. In addition, behind each potential cause would be "clues," which indicate when that cause is likely to be the true cause of the problem. A clue for the possible cause "wet pens" might suggest removing the tape and checking whether there was additional water trapped under it. These clues would be consulted by a user of the tree to help focus the search for a problem's root cause.

---

<sup>1</sup> The term "cause" is used to refer to antecedent problems. They are not necessary causes in the sense that the antecedent always leads to the problem, but rather should be thought of as prior malfunctions that may have contributed to the problem being investigated.



**Figure 1:** A simple troubleshooting tree

## 1.5. Motivation

There are three primary motivations for considering troubleshooting tools for the Corvallis production environment. First, demand for HP's TIJ pens has historically exceeded the amount that Corvallis could supply. Supply was limited not so much by line yield, but by uptime on the production lines. Some portions of downtime -- the time a production line is not producing products -- are spent implementing a fix: for example, changing a needle at an ink fill station. The rest of the downtime is spent figuring out what the underlying problem really is, and thus which fix to implement. It is this second portion of downtime that troubleshooting tools would address. If troubleshooting could be sped up, then the line would be up and running again sooner, more product would be produced in the same time, and profitability would increase.

Second, there is currently no good way of capturing information on the problems that occurred and how they were solved. The division is growing very rapidly. Engineers, operators, technicians -- most people used to know to whom to talk if they encountered an unusual problem. But today there are so many people and there is so much going on

that the informal communication networks do not function well. Since the communication channels are not working as well as they used to, problems wind up being solved multiple times, at great expense. If a tool could be provided to store past problems and their solutions, and if this tool were consulted regularly when a production problem was encountered, the organization itself would learn how to solve the new problems, rather than just the few individuals who were involved in this particular problem.

Third, there is the desire to "do more with less." On the production line the division of labor is such that operators perform routine maintenance tasks, technicians investigate problems that cannot be solved quickly, and engineers improve the processes. Troubleshooting tools would increase the scope of jobs at all levels. The machines themselves might be able to take over part of the operators' function, with the operators doing some of what the technicians do, the technicians performing some of the engineers' work, and engineers concentrating more on designing new processes.

This project does raise one question that has to be addressed under the heading of "motivation." If one can identify all of these potential problems in advance, why not just fix them? Why go to this effort to provide ways of determining when they have happened, when you have already foreseen the problems and might have been able to prevent them? Some problems are infrequent enough or expensive enough to avoid that this troubleshooting approach is justified. In addition, a tool which records and gives access to problem cause information will provide a means for growth by allowing new problems or causes that were not anticipated to be added.

## **1.6. Overview**

The rest of this thesis covers the project itself. Chapter 2 discusses current manufacturing practices at HP Corvallis. Chapter 3 presents alternative representations

for the troubleshooting information and supports the choice of a preferred representation. Chapter 4 addresses the question: For which types of problems should troubleshooting trees should be built? Chapter 5 describes the process that was used to collect the information needed to build the trees, and considers whether FMEAs (Failure Mode and Effects Analyses -- described in detail later) can serve as the information-collecting vehicle. Finally, Chapter 6 presents the conclusions drawn from this research.

## **2. Current Problem Solving Techniques**

This chapter discusses the current manufacturing environment at HP Corvallis. The product, a TIJ pen, and the production process are described. The roles of the various groups of people involved in production are presented. The chapter sets the stage for the troubleshooting project by considering what problems do occur on the existing production lines, how these problems are noticed and solved, and what tools are currently used for problem solving. Finally, the benefits of providing troubleshooting tools are enumerated in light of the specifics presented about the Corvallis manufacturing site.

### **2.1. Ink-jet pen production at HP Corvallis**

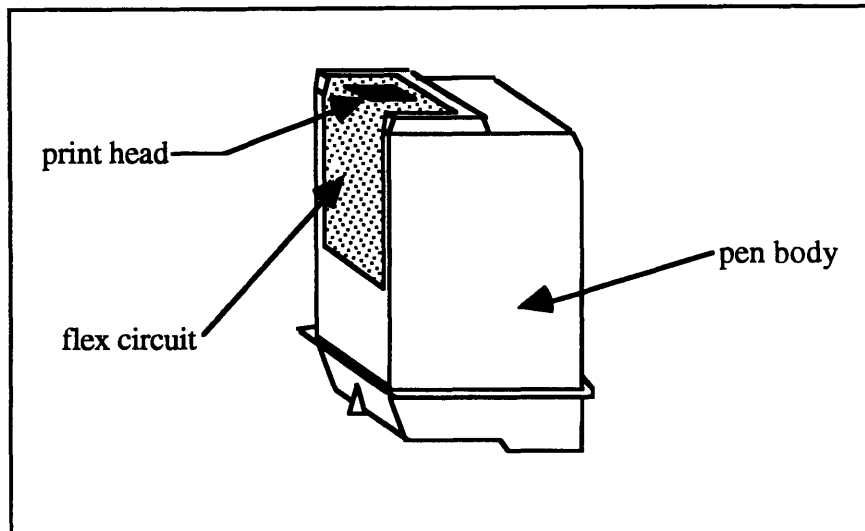
The Ink-jet Components Division (ICD) produces multiple pen products, seven of which are currently manufactured at the Corvallis site. All of the pen products incorporate TIJ technology, but they differ in the details of the pen's design. For instance, some designs use foam to create backpressure on the ink, while other designs use a spring-bag bladder for the same purpose. Certain pens hold one color of ink; others hold three ink colors.

Pen production lines likewise share high-level similarities, although the actual machines used to implement the processes differ across products and even across production lines.

#### **2.1.1. The product: A thermal ink-jet pen**

The organization which sponsored this research is designing the production line for a new product, the "M" pen. Since the product has not been introduced at the time of this writing, it cannot be described in this thesis. Instead, an older product sharing many of the same features, the "K" pen, is described.

A pen is a complex object which is used to deliver ink from the printer to the paper. It consists, at a gross level, of three major components (See Figure 2):



**Figure 2: A pen**

- The pen body is made of multiple pieces of plastic, welded together. The interior of the pen body is divided into three sections, one for each of the three colors of ink. The pen body is filled with ink and some mechanism for maintaining backpressure in the ink. In the K pen illustrated in Figure 2, backpressure is provided by three pieces of foam.
- The flex circuit, or TAB (for Tape Automated Bonding) is a rectangle of film in which are the copper traces that will allow the printer to communicate with the pen.
- The print head itself is a silicon die with channels etched into it through which the ink can flow. On top of the die sits the orifice plate, which contains the holes or orifi through which the ink will be projected onto the paper. On the die beneath each orifice is a resistor.

To print a single dot onto a page, the printer sends a signal to the pen via the connections on the flex circuit. A current is passed through the appropriate resistor on the die. Ink in the reservoir above the resistor boils, and a bubble of ink is forced out through the orifice. To print a character, multiple resistors are activated at once. The speed of the pen relative to the paper, the distance from pen to paper, the position of the ink reservoir relative to the orifice, and the shape of the orifice itself all must be finely controlled to ensure that ink from different orifi will hit the paper in consistent, predictable positions.

### **2.1.2. The manufacturing process**

The Ink-Jet Components Division encompasses four sites. At Corvallis, there are many production lines, each producing a single product. However, most products are made on more than one line. The organization that sponsored the work in this thesis is responsible for developing the manufacturing processes to produce the new M pen.

As a new pen is being developed, the processes used to manufacture it are designed in parallel with the product itself. Small quantities of the pen are produced in a prototype production area. Engineers test out processes on the prototype line, but the prototype manufacturing process consists usually of isolated machines, manually operated, with materials being moved by hand. The pens produced in this manner are tested to gather data about the performance of the product's design. The prototype line itself provides a stage for testing on a small scale the processes expected to be used on the automated production lines.

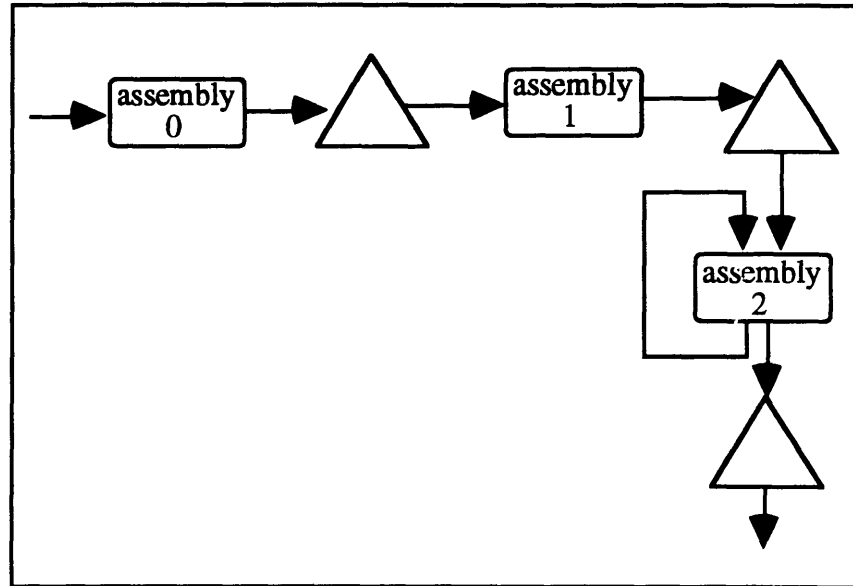
In the past, when a new product was introduced, the first line on which large quantities of the pen were produced would be a relatively slow but automated line. Then, as demand increased and experience was gained with the manufacturing processes, a high-volume line would be developed. Its cycle time would be a factor of ten shorter than the original line. The high-volume line would produce the same product, but might use



completely different machines and technologies to do so. As demand continued to increase, the high-volume line would be replicated. Therefore, the same product often was produced on different lines.

M production would proceed according to a different plan. The first automated production line to be built would be a high-volume one. Some of the slower stations would be duplicated and placed in parallel to meet the reduced cycle time goal. Ten to twenty percent of the stations on the new line would be for inspection, and five percent for audit. The line itself would be similar to the high-speed lines producing the T pen. It was hoped that this similarity would mitigate some of the risk of skipping the slower, automated line.

The M production line will be similar to most other lines on a gross level, with three major modules, usually referred to as assemblies, and large accumulators between the assemblies. (See Figure 3.) The first assembly, assembly 0, will put together the pen body and test for leakage. Assembly 1 will attach the THA (Tab-Head Assembly, consisting of a flex circuit -- the tab -- that allows communication between the printer and the pen, and a print head through which ink is directed onto the paper) to the pen body. On assembly 2 the pen will be filled with ink and print tested. Assembly 2 will contain a loop, in that pens which fail the print test will be recirculated and allowed to go through the print tester again. Pens that fail three times will be rejected. Finished pens will go into a large accumulator, from which they will be sent to packaging.



**Figure 3:** Overview of the M production line

## **2.2. Common problems on the production lines**

Problems can occur in similar ways on all of the lines. Machines fail. Accumulators fill up due to a downstream process stopping, leaving stations blocked. Accumulators empty due to an upstream process stopping, leaving stations starved for parts. Large numbers of parts get rejected by an inspection station, shutting down the line. And auditors find enough occurrences of a problem to declare all WIP (Work In Process) suspect and require a fix before allowing production to resume. One can assume that these same broad types of problems will occur on the M line as well.

Although problems can occur in similar ways, different data is collected for each of the products, resulting in an inability to compare problems across lines quantitatively. This data is presented in two ways. First, some lines (namely K and T) store production data online, accessible to everyone who knows that it is there. Second, paper reports are published about each line, as well as summary data about all lines, and distributed to management.

On the prototype M line, most of the data that is collected deals with product characteristics. Since the product as well as the processes are still under development, this feedback on the performance of actual production parts is crucial. It is circulated in paper reports. Information on problems is passed on verbally, from the operators to the engineers.

The most data collected for any line deals with the T lines. There is a wealth of process data online: downtime due to different machine faults; percentages of time that each module is up, starved, stopped by an operator, or in a machine fault state; the number of pens started and completed on each module; and reject counts from each inspection station. All of this information is extracted from the PLCs (line controllers) and so represents an actual counter value in the PLC. This sometimes leads to inconsistencies in the data, due to the way the PLCs are programmed to compute the values. For instance, downtime due to all machine faults in one report may turn out to be greater than the percentage of time spent in a machine fault state on another report. Yield figures are particularly prone to mismatch, since "yield" can be defined in so many different ways.

Data collected about the K lines is more closely related to the product than to the manufacturing lines. It includes a graph of functional defects found in audit, and information on pen characteristics.

A weekly production report is compiled which contains yields for all lines. For some lines it also lists audit failures and functional failures, and compares these to the target failure rate.

What can be inferred from the problems that have occurred on other lines about the problems likely to occur on the M line? The most detailed production data that exists is for the T line, which, fortunately, is the line most similar to the M line. Unfortunately, the largest amount of downtime has been attributed to states such as "exit track full,"

"operator cycle stop," "purge cycle completed," and so on, which do not indicate the problem on which the time was actually spent. Full exit tracks imply a downstream problem, and operator cycle stops or process stops are human-initiated and could be for any reason. There are also cases where hours were spent in an error state once, but this error state represented only small amounts of time in other months. For example, in April 1993, 151 minutes were allocated to one occurrence of a specific machine fault which never recurred in the months worth of data that were examined. Severe problems seem either to "float" from one process that causes difficulty this week, to another that causes trouble next week, or to be captured in the data under a general heading such as "operator process stop."

On the other hand, the process data may not offer insights as to what may go wrong with the M line, but the product data does offer clues as to what is likely to be a problem with the M pens. Continuous audit on the T lines found a persistent problem with nozzles not present (a specific defect in which ink does not pass through one of the pen's orifices onto the paper). Lifting flex (see Figure 2 for an example of flex) and contamination were intermittent problems, with high rates of failure one week and low rates the next. On the K lines, "lifting cheek flex", "ink under tape", and "chamber mix" accounted for the largest number of defects found in audit during the first 31 weeks of 1993. (One other defect was significant, but it tested a feature that was not relevant to the M pens, and so was ignored.)

### **2.3. How production problems are noticed**

How are people made aware that there is a problem requiring attention? This is an important question, since it is only the problems of which people are aware for which it might be worth building troubleshooting trees. There should be a signal that something is wrong, and that signal should indicate which problem has occurred, and therefore which tree to consult.

People most commonly notice that there is a production problem when a flashing light or the quiet of the line alerts them. Then they look at the dataliner screen for the module that is stopped. A dataliner is a rectangular matrix of lights, mounted overhead near each of the three assemblies, that is used to display a brief message about the status of the line. When the line is not moving, the dataliner will indicate the reason that the line is stopped. This reason may be a machine fault -- for example a part stuck in the gripper at a certain station -- or it could be an operator intervention that the line controller noticed, such as a light screen<sup>2</sup> being interrupted, or it could be a blocked line due to a full exit track after one process step.

Quality or yield problems would often be noticed by the number of parts being rejected at an inspection station, or an increasing number of defects found by the operators at the audit station. In the case of the inspection stations, once the number of consecutive defective parts exceeds a pre-set threshold, the station itself signals the line controller to shut down the module because scrap is being produced. In the case of the audit stations, the measurements taken on the pens may indicate a quality problem, in which case the auditors either interrupt the line or call a technician.

## **2.4. How production problems are solved**

Problems occur on the production lines today. How do people solve them? This section examines the current organization on a production line, and describes the troubleshooting tools that are in use today.

---

<sup>2</sup> A light screen is a beam of light just outside of the moving machinery that is used as a safety mechanism. If something gets in the way of the beam of light, such as an operator's hand, the line will stop.

## **2.4.1. People's roles in the manufacturing process**

At the production level, operators and technicians are organized around the production lines. Most lines follow the common, hierarchical organization with regard to job responsibilities, although the newest T line is a self-managed team. The lines operate 24 hours a day, seven days a week.

The production lines are almost completely automated. Production operators keep the line running; adding materials, cleaning machines, and fixing common problems. Some production operators perform visual or machine-assisted inspections. The visual inspections consist of checking every part that passes by and placing nonconforming parts on a tray in a spot reserved for the specific reason for their rejection. In the machine-assisted inspections, multiple operators pull alternating parts from the line, place them in a machine to be tested, return them to the line if they pass the test, or collect them as scrap if they fail. On the newer lines the machine-assisted inspections have been fully automated.

A process operator is responsible for each assembly, and can usually perform all of the production operator's tasks for each station in the assembly.

The third class of operator, a Process Monitor Operator or PMO, performs an audit function. PMOs remove a few parts from the line, inspect them under a microscope, and print a test pattern. If they find something wrong, they perform a crude autopsy and check for common problems. Depending on what they discover in the autopsy, they will talk to one of the production operators about the station that is causing the trouble or go straight to a maintenance technician.

The technician's job is to keep the line running. Each technician specializes in a subset of the tools on the line, so the operators have to find the right technician for the problem

at hand. Technicians also undertake ongoing improvement projects. Many of the fixes that technicians implement involve making machine adjustments or modifying PLC code to fine tune the timing.

Engineers have desks away from the production line. They fall into two groups. Process (or production) engineers are organized around products, so all the engineers working on the T product, regardless of which line, report to the same manager. But each individual process engineer is responsible for one assembly on one line. Process engineers are called in by technicians for help solving unusual machine problems. They also undertake specific process improvement projects to improve yield.

Product engineers are also organized by product, but they do not specialize in one of the separate lines that produce the product. Product engineers are concerned with the quality of the end product. They collect and publish data about pen quality, and have as a goal improving that quality. When specific pen quality problems are found, product engineers do the detective work to determine the cause.

## **2.4.2. Current problem-solving tools**

Aside from just "reasoning it out," there are some tools that people use today to help them solve problems on the production lines.

- Maintenance manuals for the machines provide some assistance in fixing problems once they have been narrowed down to a specific machine. These manuals contain a list of failure codes for the machine and the action to take when one occurs. Additionally, the PLC code dealing with that machine is included.
- SAMM is a software product that is used to track tool problems and technicians' time. It contains a history of problems for the past six months,

organized by product, by line, by assembly. Although it has fields for the symptom, solution, and downtime, the values in these fields are often not what is implied by the name. For instance, technicians tend to fill in a root cause as the symptom, rather than the symptom that first manifest itself. One specific example is "stuck valve" as a symptom. Solutions are not always provided. And the downtime is computer generated based on the time at which the report was entered, which is often well after the problem was solved.

Maintenance technicians have been asked to log their time in SAMM when they work on a tooling problem. One interview subject commented that he could not remember the last time he had done so. Technicians do occasionally consult the problem reports in SAMM when they are faced with a new problem, but there is no search capability so this is difficult. At best they search backward through the reports, to see if anything that was entered over the past week might provide a clue as to what is going on now.

- The K line has a written PMO specification for the end-of-line auditor. For each of the common defects that are found in audit, this document lists potential root causes. But it provides no information to help discriminate between the causes to find the one that is currently occurring. The causes listed in the spec are the most common ones, and the experienced technicians rarely bother consulting the document.

## **2.5. The benefits of providing troubleshooting tools**

Troubleshooting trees would provide three primary benefits. First, with the help of troubleshooting tools people could fix problems more quickly. Solving quality problems would lead to better yields, and improving uptime would result in greater utilization.



Since, in the current business environment, all pens made can be sold, both would lead to higher profits.

Second, troubleshooting trees could easily serve as a training tool, allowing new operators or technicians to investigate the possible causes of various problems on the line. Operations' staff is increasing very rapidly. One technician reported that his group had grown fifteen-fold in the past three years. But until recently there have been no training courses; people learned by following a "mentor" around for 4 to 6 weeks. This process leads to information being scrambled as it is conveyed from one person to another.

Third, troubleshooting trees would provide a repository for knowledge, keeping things that the organization has learned about past problems from being forgotten. As personnel move on to new jobs, the things they have learned about pen production will remain with the lines. Information on a peculiar problem found last year and how that problem was solved will be captured in the tool, rather than remaining only with the few individuals who participated in solving the problem.

### 3. Alternative Representations of the Information

Although the originally envisioned representation for troubleshooting trees was connections between causes and effects, other representations are possible. This chapter examines different ways of presenting the cause and effect network, as well as approaches that do not involve explicit causes and effects at all.

First, though, what is meant by "troubleshooting" must be considered. For the purposes of this research in the context of a high-volume production line, troubleshooting involves identifying the root cause of an observed problem and describing a solution to that root problem. In the Computer Science or Artificial Intelligence (AI) field, the process of identifying a root cause of a problem or finding the fundamental malfunction in a system is referred to as *diagnosis*.

Several techniques have been developed for solving diagnostic problems. Among these are:

- Model-based reasoning
- Case-based reasoning
- Decision trees
- Bayesian updating

A problem-solving system using any of these techniques has two fundamental components: a representation for the knowledge or information; and a decision procedure for utilizing the knowledge to find a root cause and/or a solution.

In the two sections that follow, the advanced diagnostic reasoning techniques mentioned above are considered first, and simpler representations of cause and effect are considered second.

### **3.1. AI representations for diagnostic problem solving**

In AI implementations of diagnostic systems, the knowledge representation and diagnostic procedure are both implemented in software. A knowledge engineer gathers information about the problem and determines an appropriate representation for that knowledge. The knowledge engineer collects more information about the problem, often by interviewing "experts," and encodes that knowledge into the chosen representation. Occasionally tools are provided to allow the experts themselves to encode their own knowledge.

The knowledge representation may suggest a certain decision procedure. For example, using a decision tree to encode problem knowledge implies an ask-a-question-follow-the-branch-of-the-tree-corresponding-to-the-answer procedure for getting to the root cause. The knowledge engineer writes a program implementing the chosen decision procedure. More and more, rather than writing their own programs, knowledge engineers are using off-the-shelf tools which have decision procedures already built into them and incorporate multiple knowledge representation paradigms.

#### **3.1.1. Model-based reasoning**

A model-based approach to problem solving is based on "the methodical enumeration and relaxation of underlying assumptions about the device."<sup>3</sup> The model itself usually consists of two components: structural dependencies, such as the connections between

---

<sup>3</sup> "Diagnostic Reasoning Based on Structure and Behavior," Randall Davis, *Artificial Intelligence*, V. 24, 1984, p. 407

components; and a functional model describing how the output of each component depends on its inputs. The reasoning process proceeds by gathering a description of the actual inputs and outputs of the system. It then simulates the system, using the model to propagate values for the actual inputs through to establish what the expected outputs would be. Starting with the places where the actual and expected outputs differ, reasoning proceeds “backwards” through the model to figure out which model assumptions, if relaxed, would remove the inconsistency between the expected and observed outputs. If a simulation with the relaxed constraints removes the inconsistency in outputs, then that constraint represents a possible cause of the failure.

To apply model-based reasoning to problems on a pen production line, one would create a functional model of each of the processes, concentrating on the way movement through the process/station changed both the pen and the state of the station. The structural model would describe how the stations were connected together.

There are large difficulties in applying model-based reasoning to a production line. According to Davis and Hamscher, the complexity lies in building the model: “Diagnostic reasoning from a tractable model is largely well understood... There is, by contrast, a rich supply of open research issues in the modeling process itself.”<sup>4</sup> The number of stations on the line (on the order of 100) does not in itself make the problem too large for a model-based diagnostic solution. Rather, the complexity of the stations and the fact that the values to be propagated would be symbolic rather than numeric combine to make the problem difficult.

There are additional reasons that model-based reasoning may not be appropriate for troubleshooting of a production line. Some stations have “memory” -- for example,

---

<sup>4</sup> "Model-based Reasoning: Troubleshooting," Randall Davis and Walter Hamscher, chapter 8 in Exploring Artificial Intelligence, by Howard Shrobe *et al*, p. 298

counting pens and offloading every fourth one to a different bonding machine. Since the model is run backward as well as forward, behavior must be invertible. Complex functions with state (memory) are often not invertible. Additionally, this approach to diagnostic problem solving assumes that there is a single point of failure. Cases where two things combined to cause a problem -- plastic parts from incompatible molds, combined with a poor weld -- would not be addressed.

### **3.1.2. Case-based reasoning**

Using this technique, knowledge is represented in the form of cases, which are stored in a database. A case includes a problem report, a solution, and indices describing the salient aspects of the problem. The decision procedure consists of obtaining a description of the new problem, using similarity metrics to retrieve a case that describes a problem like the current one, possibly modifying the retrieved solution to fit the particulars of the current problem, testing the new solution and repairing it if necessary, and finally storing the problem and its solution as a new case.<sup>5</sup>

There are two primary differences between case-based reasoning and a simpler dictionary of problems and their solutions. First, a case-based reasoning system goes beyond retrieving the solution to a problem that is an exact match to the current problem, and can modify a retrieved solution to fit a problem that is similar to, but not exactly the same as, a previously known problem. Second, a case-based reasoning system automatically learns and remembers new cases. As it modifies old solutions to fit new problems, it stores the new problem and newly created solution as a new case.

The database in which maintenance technicians store problem reports, SAMM, would be a logical starting point for creating a case repository. However, it often does not contain

---

<sup>5</sup> "Case-Based Reasoning: A Research Paradigm," Stephen Slade, *AI Magazine*. 1991, p. 45 -- 46

descriptions of the *problem*, just of the solution. Furthermore, there are no indices into the “cases” in SAMM. Problems for which maintenance technicians are not called would have to be added on their own. And along with collecting and representing the case knowledge, the decision procedure would have to be written.

### **3.1.3. Decision trees**

A decision tree is similar to a troubleshooting flowchart. Knowledge is represented as a test or question, followed by paths to follow for each possible result or answer. On each path is another test or question, followed by *its* paths. This knowledge can be made visible to the user in a graphical format, with tests as the nodes and possible answers labeling the edges. Or the tree representation can be stored internally on a computer, and never presented visually. In either case, the decision procedure is to ask the question (perform the test) at the top of the tree, compare the result to the labels on each of the paths leaving the top node, follow the path that matches the user's answer, ask the question at the next node on the path, *etc.* The leaf nodes of the tree would correspond to root problems, each with a recommended solution. When the decision procedure reached the end of a path, it would advise the user to perform the fix indicated in the last node on that path.

To create a decision tree for a particular diagnostic problem, the problem must fit the decision tree structure. To be more specific, there have to be tests that can consistently discriminate between paths to follow and narrow down the set of possible root causes. Most often this means that there is some hierarchical structure to the potential problems. A common example would be diagnosing a system with subsystems, say a car. Is there any test that would help you determine whether the fault lies in the electrical system, *vs.* the fuel system, *vs.* the exhaust system, *etc.*? In most complex problems -- for example, medical diagnosis -- there are so many exceptions or problems in one system that can mimic problems in another system, that this simple decision tree paradigm breaks down.

### 3.1.4. Bayesian updating

Bayes theorem provides a mechanism for inferring causes from the effects that are observed. To do so, it uses probabilities as its underlying knowledge representation. The following are required:

- The *a priori* probability of each effect occurring given that each cause is present.
- The *a priori* probability that each cause is present.
- The *a priori* probability that each effect is present.

Bayes theorem, a formula for computing the probability of each of the causes once a value for each of the effects is known, provides the decision procedure. With slight additional work, the decision procedure can be amended to calculate the value of knowing about each remaining effect, compare that to the cost of obtaining a value, and optimize the amount of information that is requested. Once one cause is assigned a sufficiently high probability, it is presented as the most likely to have occurred.

One overriding shortcoming of using this technique for troubleshooting is the amount of data that must be provided. If there are  $N$  possible root causes and  $M$  possible effects (observed problems), then  $O(M*N)$  probabilities must be collected. The simple formula only applies if all of the effects (observed problems) are statistically independent of one another. If this is not the case, even more probabilities must be collected.

## 3.2. Representations of cause and effect

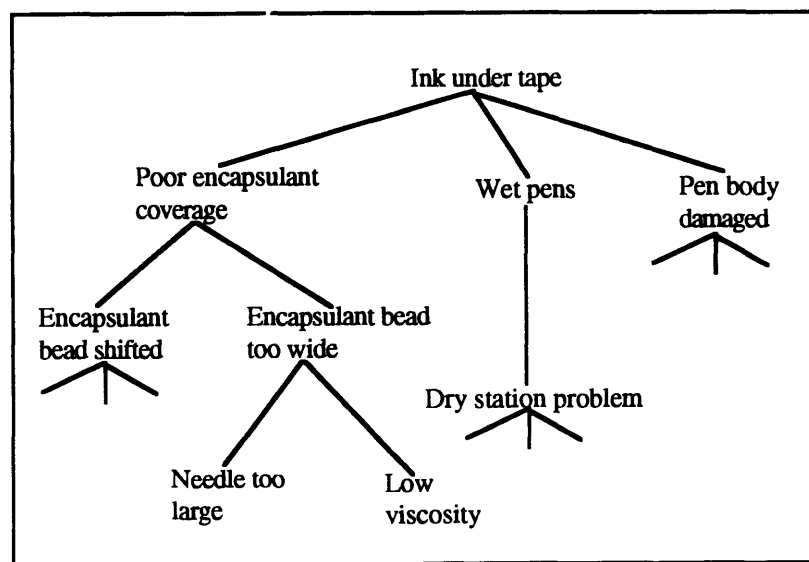
In contrast with the AI representations presented in the previous section, the cause and effect representations of troubleshooting knowledge described here do not incorporate a

decision procedure. Rather, the user is expected to provide this procedure by investigating the causes and effects in an order that makes sense for the problem at hand.

In one sense the knowledge representation for all of the options presented below is the same: connections between effects, their causes, their causes' causes, and so on. But given this underlying, conceptual representation, there remains a choice of how to present the cause and effect network to the user. It is the user's view of that representation that is considered in this section. Three alternatives are considered below.

### 3.2.1. Graphical

The original formulation of the troubleshooting tree concept was as a graphical display of information similar to that created for a fishbone diagram or by using the "ask why 5 times" technique. As the causes and effects were collected, flowcharting or graphing software could be used to display the causal connections in a format similar to a family tree. (See Figure 4.)



**Figure 4:** A troubleshooting tree displayed as a graph



Troubleshooting trees would be plotted on large sheets of paper and hung near the production line.

This concept quickly became unwieldy as the number of nodes in a troubleshooting tree grew too large. Furthermore, it did not provide a good way to reuse lower portions of a tree that were common to several problems. No additional information was provided on how to tell which path to follow through a tree. And solutions would have to be listed separately, indexed by root cause, if they were not immediately obvious by identifying the root cause.

### **3.2.2. A database with a forms interface**

Problems and their causes could be stored in a database. Every cause would also be a problem, with its own set of causes. If there were no causes for a problem, it represented a root cause, and should have a solution. When a problem was noticed on the line, the user would select the current problem from a list, and *its* causes would in turn be retrieved and displayed. This would continue until, instead of a list of causes, a solution would be retrieved.

In addition to storing a list of causes or a solution with each problem, "clues" could also be stored. A clue would provide information to help determine whether this particular problem was likely to have occurred. In some cases, a clue might pose a question which would rule in (or out) a particular path of inquiry. For example, droplets of ink on the orifice plate are more likely to have been caused by water on the orifice plate if they occur in a certain position. In cases where no test exists, broad estimates of the probability of a cause occurring might be included. Thus, the user could consult the clues to hone in on the most likely cause to pursue.

Interfaces would be provided to the database both to search (find the root cause of a problem) and to add new information. New information could take the form of newly

discovered causes of a problem, better clues, or even a tree for a completely new problem.

### **3.2.3. Hypertext**

Hypertext is similar to the database representation presented above, except that more of it comes in an off-the-shelf package. The structure of the underlying information is hidden from the person creating the application. Additionally, functions are provided for easily building a specific type of interface to the structured data.

Hypertext uses cascading menus or "hot" areas of text and graphics to link areas on one screen with other screens. When a user clicks the mouse on a "hot" area, a screen associated with that area is brought up and displayed. The hypertext system itself manages the mouse and screen interactions; the programmer merely enters the text or graphics, indicates which areas are "hot," and establishes the connection between a "hot" area and its subsequent screen.

A hypertext implementation of troubleshooting trees would look very similar to the forms interface described in conjunction with a database. The user would be presented with a list of problems, click on one, see a list of causes, click on one, see a list of *its* causes, *etc.* Solutions would be the screens reached by clicking on root causes, and would have no "hot" areas themselves. Clues could be provided by listing them along with the causes, but not making them "hot."

### **3.3. The recommended representation**

A hypertext format was chosen for a prototype of the troubleshooting trees for the M production line. The hyperlink feature of Interleaf was selected as the implementation vehicle. Reasons for this choice were:

- Users of the troubleshooting trees are reasonably knowledgeable about the problems that do occur and which causes are more likely. Thus, there was not deemed to be a great need to have the system itself do all of the reasoning. Therefore, the cause and effect representations were preferable to the AI representations, since their implementation entailed significantly less effort.
- There was no resource identified to take on the "knowledge engineer" role. Since it was desired that users be able to extend the system, no representation that required advanced training to extend was suitable.
- Interleaf was recently adopted as the standard vehicle for communicating production documents. All production workers will eventually be trained in navigating through an on-line Interleaf document and using Interleaf hypertext applications. Furthermore, the troubleshooting trees and the maintenance manuals will be interlinked, with a troubleshooting tree often referring to a picture in a maintenance manual to illustrate a problem. The ability to store the shared information in one place and refer to it within the maintenance manual and from the troubleshooting trees will facilitate the creation and maintenance of the trees.
- The Interleaf product runs on Unix workstations, and can also be accessed through a PC. There is already some operator familiarity with PC applications, and unix platforms leave room for choice in future production lines.
- It is easy to build a prototype application quickly with a hypertext tool. The "knowledge base" of problem causes can be built up incrementally, and feedback can be gathered before the system is complete.

- In the end-user environment, Interleaf provides support for adding "notes" to any portion of the application. This feature is ideal for allowing end-users (operators as well as engineers) to add comments indicating new causes, tests that they found useful for determining which cause to follow, or other matters. Before these notes were incorporated into the main body of the application, they could still be viewed by other users.

## **4. Selecting a Subset of the Line for Which To Build a Troubleshooting Tree**

Troubleshooting tools are not needed for every problem that could occur on the production line. Some problems can be solved up front, by redesigning a station to eliminate a failure mode. Other problems are so common and their solution so obvious (for example, part jams), that no tool is needed to help identify a remedy. Still other problems will be unique, and so not included in existing troubleshooting procedures. Add to this the idea of incremental development and the benefits of providing a tool to be used before one is complete, and it becomes clear that the troubleshooting tool should be usable before it has addressed all of the problems that could occur on the production line.

Given that troubleshooting trees will not be built for the entire line, or at least will be built incrementally, the question of where to start arises. This question is really asking for a definition of a *problem*. For what kind of problems does it make sense to build troubleshooting trees? For a start, troubleshooting trees should be built for events that will give production operators a clear signal that there is a problem and that they should begin investigating.

### **4.1. Possible "roots" for a troubleshooting tree**

If troubleshooting trees will not be built for every problem that might occur on the production line, what is the best type of problem for which to build these trees? What category of production problem should serve as the roots of the troubleshooting trees? Although the specific criteria used to select the best type of problem are presented in detail later in this chapter, one criterion was used to generate the candidates and deserves mention at this point. The first criterion is that the troubleshooting trees have to

correspond to problems that are observable on the line. The problem that is noticed should point someone towards one tree. So the roots of the trees should be something that people see and recognize, and for which they have a name.

In the sections that follow four proposals will be considered around which to build troubleshooting tools. These four areas include:

1. Failures identified at an inspection station
2. Defects found at an audit station
3. Problems with a specific machine or process
4. Messages displayed on a dataliner

The remainder of this section describes in detail each of these four types of problem. The next section presents and justifies the criteria that will be used for selecting the best among the alternatives. In the final section the alternatives are evaluated according to the selection criteria, and the type of problem best suited for the troubleshooting tree format is chosen.

### **4.1.1. Failures at an inspection station**

There are over twenty inspection stations on the M production line. Most of these just indicate pass/fail, although some have more detailed categories of failures. One complex inspection station, the Automated Print Quality Tester (APQT), checks multiple operations of each nozzle/resistor combination on the pen, and so can indicate different types of failure for any of the nozzles. Certain inspection stations test electrical functioning of the pen/flex circuit at different stages during the assembly process. This type of test, too, can indicate different failure codes. Although the failure codes are stored in a production database and used for after-the-fact analysis of problem trends,

they do not by themselves indicate that there is a production problem. Rather, a problem that bears further investigation is signaled when the inspection station notices an out of control station. That is, SPC rules such as "seven in a row above the mean" are what is really used to indicate a problem at an inspection station.

A pen that fails inspection does not itself mean there is a production problem. The processes cannot be controlled finely enough to be expected to produce no bad pens. But failure modes, in combination with inspection stations exceeding an SPC-set threshold (*e.g.* more than 2 pens in a row had the flex attached to the pen crookedly) do indicate problems, and could serve as the roots of troubleshooting trees. The line shuts itself down when it counts too many of these inspection station failures, so operators would certainly be aware of the problem and know to start investigating for causes. But how often do inspection-related failures shut down the line? More often, at least from the historical data for the K line, the lines do not seem to shut down for these problems. Instead, when aggregate production data is compiled, Pareto charts indicate a clear quality problem *after the fact*.

### **4.1.2. Defects at an audit station**

The M production line will have an audit station at the end of each of its three major assemblies. There is also an Finished Goods (FGS) audit station, which will receive completed, packaged pens hours after they have been built. For in-process audit on the M line, a continuous audit process will be used. Every  $n$ th pen will automatically be shuttled off the line and into an audit station, where it can be analyzed. With the continuous audit process, the sampling rate will depend on the number of defective pens found in recent sampling periods. Thus, as more pens are found to be defective, more will be removed from the line to be inspected. Eventually, the queue into the audit station will fill up, blocking the rest of the line and causing it to stop. So the audit process itself can shut down the line.

At the audit station, an operator examines individual pens. This can involve visually inspecting the pen, characterizing it in terms of certain non-functional features such as weight, dissecting the pen and characterizing the destructive forces necessary to break the bonds, and performing a functional test and assessing the print quality. Defect codes are assigned for measurements out of spec and other cosmetic or functional failures.

The FGS audit is so far removed in time from production that any defects identified there could result in hours of inventory that had to be discarded. But a primary purpose of continuous audit is to provide early feedback on the production processes. Thus, the auditors themselves are required to do some detective work when they identify a defect, tracing the problem back to its root cause and notifying the appropriate process operators. Troubleshooting trees built around defects found at the audit stations would support this analysis. Speeding up the identification of the root causes of defects would result in both fewer defective pens (because fewer pens would be produced in a shorter time) and higher line uptime (since, with the continuous audit process, more defects leads to more pens removed to be audited, which would eventually create a queue long enough to block and stop the rest of the line).

### **4.1.3. Processes**

Most problems can be traced to a specific process or station on the line. If, for example, the encapsulant is found to be off target, one would suspect the encapsulant dispense machine. Additionally, the manufacturing development engineers who design the tooling for a line have been encouraged to perform potential problem analyses for their stations. The potential problem analysis considers ways that the tool could malfunction, and identifies the effects that would result from each malfunction. The failure and effect information taken from these analyses could be used for troubleshooting. But could troubleshooting trees be organized with processes as their roots?



One could certainly create a troubleshooting tree for problems related to a specific process, for example that of filling a pen with ink. But having a separate tree for each process leaves to the user the task of determining which process is at fault and, therefore, which tree to consult. A tree built around a process would be similar to the lower branches of a tree built around a fault.

#### **4.1.4. Dataliner messages**

A dataliner is a rectangular matrix of lights that is used to spell out brief messages about the status of the production line. The line controller, a computer, governs which messages are displayed and causes them to be posted on the dataliner. Each of the three assemblies has its own dataliner which displays messages unique to that assembly. Most of the hundreds of dataliner messages represent problems that are occurring on the line. These messages themselves could well serve as the roots of troubleshooting trees, with the tree providing the possible faults that could lead to the message.

#### **4.2. Criteria for selection**

This section presents a description of the ideal problem for which to build a troubleshooting tree. The characteristics of the ideal problem will then become the criteria for choosing among the four candidates identified above.

First, there should be a name that is used to refer to the problem and some means used to alert people that the problem has occurred. If there were no name, then when people looked to the troubleshooting trees for help finding the root cause of the problem, they would not know with which tree to start. They would have difficulty just finding the applicable tree for their problem, not to mention traversing the tree's causes and locating the problem's root cause and solution.

Second, the problem should take a while to solve today. Additionally, a significant portion of the time to solve the problem should be spent on detective work; tracking down the real root cause. The rest of the time will be spent implementing the solution. It will take a while for the user to traverse the troubleshooting tree and find the root cause of a problem, and if using the tool does not reduce the time it takes, few people will use the troubleshooting trees.

Third, there must be people or other sources of information that can identify the majority of the possible root causes of the problem. If these people did not exist, there would not be a way to get the information to place in the tree. The cause and effect information must come from somewhere.

Fourth, most potential users of the troubleshooting trees should not already be able to identify the root causes of the problem. If most people already knew the causes, a troubleshooting tree might serve to jog their memories for causes which they have forgotten, but it would not give them much new information and so would be of little value.

Finally, if several problems meet the first four criteria, the problem that occurs most often should be chosen. Something that accounts for a large amount of downtime or yield loss would have higher leverage and make the best initial display of the concept of troubleshooting trees.

### **4.3. Comparison of the alternatives**

This section evaluates the four candidate roots for troubleshooting trees using the evaluation criteria set forth in the previous section. Where possible, data from the K and T lines are used to justify eliminating or recommending one of the alternatives.

### **4.3.1. Failures at an inspection station**

It is difficult to evaluate the applicability of troubleshooting trees to the abstract idea of inspection station failures. The different inspection stations catch quite different product failures. Instead of evaluating the appropriateness of inspection failures in general, specific candidate inspection stations were generated. Engineers designing the M line were asked to name problems likely to occur on the line that would have many possible causes and require a long time to track down the root cause. Four inspection stations were mentioned. These were:

- APQT, an Automated Print (or Pen) Quality Tester: The APQT machine on the M line will detect over a hundred failure modes, which represent about 15 different failures, some of which can occur for any nozzle on the pen. Examples of failure modes are "(nozzle) not present," "edge rough" or "deprime" for each of three colors.
- Weld problems, detected by the leak testers: The leak testers will find problems primarily with the plastic materials or with the welding process. Welding is a particularly difficult process, and was thought to offer good opportunities for troubleshooting tools. The output of the leak tester is ml of air per chamber of the pen. This really gives rise to only one tree -- for leaky pen bodies.
- Head alignment: This inspection station checks the orientation of the tab with respect to the pen body. The output of the inspection station is likely to be "Target out of spec" or "Vision system problem," although the output has not been formally defined yet.

- E-test (electrical test): There are several E-tests during final assembly, and another one in the bonding process that occurs prior to final assembly. The E-test considered here occurs at the end of assembly 1, and thus can detect electrical failures that have been introduced during the tab attach processes. There will be around fifty different E-test failure codes for pens -- open, short, leakage, high and low resistance -- each of which can specify the primitive that is failing. A troubleshooting tree would be built for each of these five classes of E-test failure.

Each of the four candidate inspection stations is evaluated separately.

The APQT station does produce named problems: its failure modes. Problems found at the APQT station take time to solve. But it is not clear whether there are people who can identify the causes of APQT problems. One engineer who had tried to track down the root causes of APQT problems for an older pen product, S, believes that many of the failures are "TIJ mysteries" related to interactions between the ink and the thin film of the print head.<sup>6</sup> Few people know the causes of APQT failures, so troubleshooting trees for these problems would certainly be useful. APQT failure modes would be good roots for troubleshooting trees in terms of usefulness, but it is not clear that the knowledge exists to put into the trees.

There is currently no mechanism planned for signaling a leak test problem other than direct observation of rejects at the leak testers by the operators. Pens that leak will be offloaded, but no alarms will go off if too many pens fail. In addition, there is already a means for solving leak problems which involves correlating leaky pens to both welder

---

<sup>6</sup> Interview with Tim Hubley, 8/3/93.

and tester.<sup>7</sup> Troubleshooting trees do not fit leak test problems, since there is already a simpler means for tracking down the root cause.

Head alignment initially appeared to be a good area for building troubleshooting trees. Yield loss due to misaligned heads on the T lines is significant., and tolerances on the M line will be twice as tight as on the T line. At first glance this indicates the possibility of a high payback for improving head alignment. The dominant fix for alignment problems is to adjust the initial placement offset, and the most common problem is a shift during curing. In addition, the placement and cure processes are still being changed, and so the same solutions to an alignment problem will not be available to the operators once the line is up and running as are available to the engineers today. Operators and technicians can only adjust the initial displacement or perhaps fix staker shoes. A troubleshooting tree would not be able to recommend many different solutions. The need is more for some sort of feedback control from the alignment tester to the placement machine.<sup>8</sup> Troubleshooting trees are not appropriate for head alignment problems, since there is one overriding solution.

E-test assigns failure codes to rejected pens, but signals an error (and stops the line) only when more than three pens in a row fail. Since parts passed an E-test earlier in assembly 1, failures identified at the station under consideration would most likely be due to one of the four intervening processes damaging the flex or the orifice plate. The engineers designing these four stations could probably identify most of the ways in which E-test failures might be introduced. Finally, although E-test rejects accounted for a significant percentage of the pens started on the T lines, failures on the M line are expected to account for a much smaller percentage, due to an E-test earlier in the process which

---

<sup>7</sup> Interview with Jim Clark, 8/19/93

<sup>8</sup> Interviews with Melina McCarty on 8/18/93 and Craig Olbrich on 8/23/93

should catch many of the problems before they progress. Therefore, failures significant enough to be noticed (more than three in a row) are expected to be quite rare.<sup>9</sup> E-test failures are considered to be inappropriate roots of troubleshooting trees, since failures will likely be rare enough and be solved quickly enough that troubleshooting tools will not be needed.

### **4.3.2. Defects at an audit station**

The End-Of-Line (EOL) audit station was investigated as a possible producer of problems for which to build troubleshooting trees. The defects found in each hour are graphed on a control chart, and a problem is indicated when the latest point exceeds a control limit. The audit checklist includes 11 cosmetic defects, 17 visual defects, 14 functional defects, 7 print test functional defects, and 5 print quality defects. The knowledge does exist within the organization of causes of these defects, as evidenced by written procedures for the EOL auditors on the K line documenting possible causes of the defects. Additional causes were mentioned in the Failure Mode and Effects Analyses (FMEAs) done by the manufacturing development engineers, so it is likely that interviews would elicit many more potential causes. Some problems take a long time to solve, as indicated by the continuation of a large number of defects in the Functional Defect Report from week to week. The operators already know some of the causes, since they are listed in their work specification. But the specification says to call engineers and technicians at certain points, so these points would be good places to create troubleshooting trees and solve the problems closer to the line.

---

<sup>9</sup> Interview with Ted Barnes, 8/18/93

### **4.3.3. Processes**

Although most problems can eventually be traced to a process problem, processes are not themselves good roots for troubleshooting trees. The problems should be indexed in some other manner, and may lead eventually to processes as causes. There is no clear signal that a particular process problem exists. Finding the right troubleshooting tree to help solve a problem would itself be a challenge.

### **4.3.4. Dataliner messages**

Dataliner messages for the M line have not been defined yet, but many of them can be predicted. Once they are defined, there will be hundreds of messages. For the most part, they will bring the faults that the line controller (a PLC) notices to the attention of the operators. Messages will be for things such as "Part jam station 21" or "Low materials station 1" or "Exit track full." While the causes of these situations might not be known (Why did the part jam?), the solutions are usually quite straightforward. (Remove the jammed part!) And getting at the solution is the real purpose of the troubleshooting tree.

Data exists for the T line on which PLC fault conditions (analogous to dataliner messages) occurred during a given time period, the number of occurrences, and the cumulative downtime due to each fault. Four months worth of data on two of the T production lines was searched for faults in which the MTTR (mean time to recovery; the cumulative downtime divided by the number of occurrences) was greater than ten minutes. These faults are presented in Table 1.

<b>Fault</b>	<b>MTTR</b>	<b>Fault</b>	<b>MTTR</b>
Sta 16 cross shuttle jam	30.9	Exit track full	13.6
Operator process stop	24.9	Purge cycle completed	127.2
Upper guards bypassed	16.2	Sta 1 pen body not in place	948.0
Sta 11 screen not in place	35.0	Sta 8 pick & place o/l	19.8
Sta 37 anvil not down	11.7	Sta 42 anvil not down	19.0
Sta 14 flex align h/w error	23.4	Sta 4 struct vision h/w error	19.8
Sta 1 rotator jam	15.0	Sta 14 preload separator vac	22.0
Sta 20 prime separator vac	99.0	Sta 27 gripper 4 overload	18.0
Sta 31 turnover placement	17.3	Sta 34 temp out of range	16.3

**Table 1: PLC faults with MTTR greater than 10 minutes**

Two caveats must be included with Table 1. First, in the raw data from which this table was extracted, faults are listed with non-zero durations and zero occurrences, which is clearly nonsense and makes the data suspect. Second, several of the faults with overall MTTR greater than ten minutes were dominated by only one occurrence of the fault.

The caveats notwithstanding, one can conclude that dataliner messages are not appropriate problems to serve as the roots of troubleshooting trees. First, the MTTR in most cases is rather short. Most problems can be resolved within 20 minutes. Second, some faults for which the MTTR is large do not really indicate what the problem is. When the dataliner flashes "Operator process stop" or "Upper guards bypassed" or "Exit track full," the messages give almost no clue as to why the line is stopped. Yes, an operator may have opened the guards around a machine, but that was probably in the process of solving an existing problem. Third, some of the faults with high MTTRs are so specific that their cause would probably not be found by troubleshooting trees. A fault such as the temperature being out of range or a hardware error with a vision inspection system would most likely be the lowest level in a troubleshooting tree.



## **4.4. The chosen problem**

On the basis of the analysis presented above, the EOL audit station was chosen as the best place for which to build troubleshooting trees. The purpose of the audit station is to provide quick feedback to the process on its performance. This makes it a natural position at which problems will be investigated. EOL auditors in particular are charged with tracking down root causes of defects. For these reasons, end of line audit was selected as the area for which to build troubleshooting trees.

A troubleshooting tree will be built for each of the defects identified in the EOL checklist. The problems of "ink under tape" and "cheek flex lift" were chosen as the roots of the first trees. Lifting flex on the cheek of the pen has been a consistent problem on the K line, usually accounting for the largest number of defects. Ink under tape, while a frequent problem, was chosen more because it was thought to be a complex problem, whose causes would also be causes of other problems, and so building a tree for it would test the limits of the troubleshooting tree concept and simplify the task of building the next few trees.

## **5. Collecting Problem Solving Information**

This section presents a procedure for gathering the information that is needed to create a troubleshooting tree. First it lists the information that is needed to create a troubleshooting tree. Then it describes FMEAs, an analysis procedure that engineers currently follow, and considers whether the FMEAs can provide the necessary information. Finally it presents a procedure for collecting the information.

### **5.1. The information required to create a troubleshooting tree**

In order to create a tool to assist in finding and solving the root causes of production problems, information of the following sort is necessary:

- Connections between problem causes and their effects are needed. Ideally, this connection should be between a cause and its immediate effect, not the end effect. This link to immediate effect, then from that effect to its subsequent effect, and so on, will provide for a tree structure to the possible cause information. This tree structure will both enable information to be reused when multiple effects could have the same cause, as well as make for much easier searching through the possible causes.
- Effects should relate to what is observable in production. People will have to notice problems. And once they notice them, if they are to use the troubleshooting trees, the problem that they notice should provide an entry point into the trees. Effects that are visible after production is finished, while important effects, are less useful for troubleshooting purposes.

- The terms used should be consistent, both so that they can be linked, and so that a user can identify a problem in common terms and thus begin searching for its causes.
- The majority of causes for a given problem should be included. If a troubleshooting tree includes 80% of the causes for each problem it covers, it is likely to be useful to people trying to track down root causes. If, however, it only covers 20% of the causes, the usefulness of the tool is questionable. It is not clear exactly where the line between "enough" and "not enough" falls, but once the coverage passes the boundary, it may be that the trees are useful enough that they will be self-perpetuating. If people find them to be valuable, they will be more likely to add new causes that they encounter to the trees.
- Root causes should have solutions attached to them.
- "Clues" as to when a cause is applicable would be helpful. Each cause should have an indication of when it is likely to be the correct cause. Perhaps default probabilities from a Pareto chart would be a start towards this, but something to look for that would either rule out this cause or indicate that it is a very likely suspect would be ideal.

## **5.2. FMEAs as a source of troubleshooting information**

This project initially looked at using FMEAs as a source of information because the FMEAs were already being created for another purpose, and so the troubleshooting information came for free. In the following sections, FMEAs are described, and how they were intended to be used to create troubleshooting trees is explained. Then their appropriateness as a source of troubleshooting information is evaluated. For more information on FMEAs, please see the reference included in the bibliography.

## 5.2.1. Description of FMEA

The creation of an FMEA begins by identifying the function that a process is intended to perform. Once the function has been identified, the ways the process can fail to perform the function -- called failure modes or potential problems -- are listed. Assume, for example, an ink fill process whose function is to fill a pen with ink. (See Figure 5.) Possible failures would be not putting enough ink in the pen, filling the pen too full, getting ink on the outside of the pen, *etc.* For each failure mode, possible causes and customer-visible effects are identified. Each failure is then ranked according to its frequency of occurrence (O), difficulty of detection (D), and severity (S). For failures with high aggregate priority (RPN, the product of O, D, and S), either the design is changed to eliminate the problem, or control actions are recommended. These are actions to reduce the likelihood of the failure, not to correct the failure if it should occur. The entire table of potential failures is reviewed with a group including engineers working on other processes.

Function: Fill the pen with ink.

Failure	Cause	Effect	O	D	S	RPN	Recommended Action
Insufficient ink in pen	Clogged filter, Low ink in reservoir, Clogged needle	Short pen life, Low pen weight	2	4	5	40	Weigh pen after filling, Use special needles

**Figure 5:** Portion of FMEA for ink fill process

The FMEA could be turned into a troubleshooting tree, as illustrated in Figure 6. Effects are linked to causes by failures. The leaf nodes at the bottom of the tree represent root causes, each of which should have a solution associated with it. If more failures had

been identified there might be multiple paths to the same root cause, as illustrated with "Clogged needle," or paths from one effect to different root causes, as illustrated with "Short pen life." If FMEAs were done for all of the other processes involved, the tree would be deeper, since causes of a failure in one process may be effects of failures at upstream processes. This is illustrated by "Low pen weight."

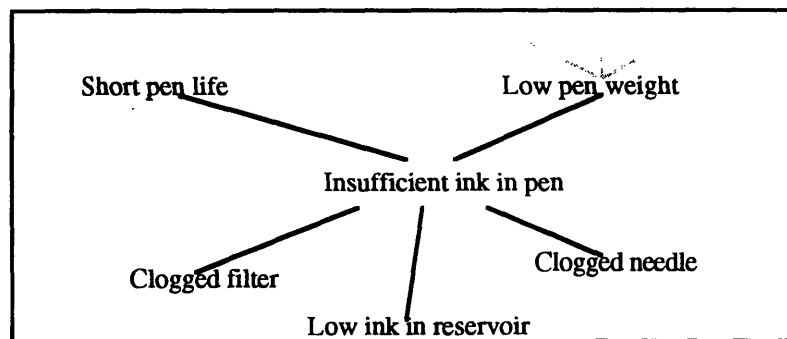


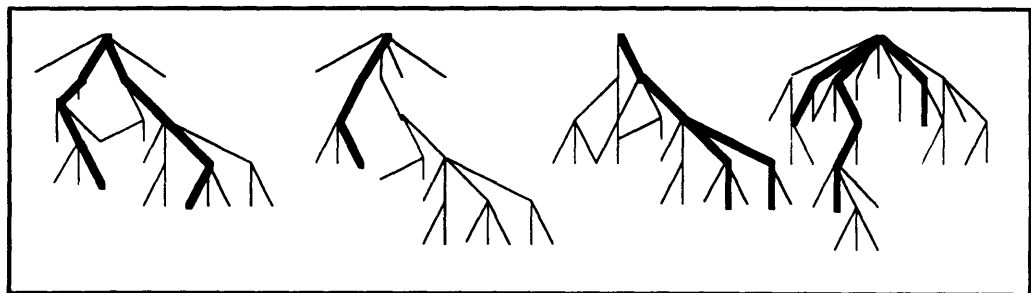
Figure 6: Fragment of a troubleshooting tree

## 5.2.2. Appropriateness of FMEA

While FMEAs contain a good deal of information that may be useful for diagnosing problems on the production line, they are not suited to being the primary information gathering medium for creating troubleshooting trees. Reasons that FMEAs are inappropriate include:

- Individual FMEAs have broad, sparse coverage. The ultimate effects of the failures identified in an FMEA are likely to be scattered across the problems that a user would observe. This is illustrated in Figure 7. Each tree represents the potential causes of a different defect. The bold lines indicate the subset of all possible causes that are due to one process, and so would have been identified in the FMEA for that process. The light lines represent the other possible causes, due to other processes. Consider someone who is using an

early version of the troubleshooting trees. FMEAs have not been done for all processes, so only some of the paths through the trees (potential causes of problems) have been encoded. No matter what problem the user is investigating, only a few possible causes have been identified. The user will probably not find the troubleshooting trees very helpful and soon stop consulting them.



**Figure 7:** Portions of troubleshooting trees identified by a single FMEA

- FMEAs assume that incoming materials are good. This leads to their capturing only single failures. Any failures that require two things to go wrong, or that compound previous failures, would not be identified by the standard FMEA creation process. This means that often only pieces of the cause and effect links are captured in an FMEA. The fact that a high encapsulant bead could lead to taping problems, for instance, was included in the encapsulant FMEA. But the taping FMEA did not continue with that line of reasoning and indicate what particular kind of taping problems would occur or what the effect of those problems would be. To gather the additional information to create *trees*, as opposed to uncompleted *branches*, requires further discussion with the engineers who prepared the FMEAs.

Another consequence of assuming that materials are good is that defective materials are not included in the troubleshooting trees. Through interviews with maintenance technicians on the K and T lines, it was learned that material problems account for more than half of the problems on the lines.

- FMEAs do not identify solutions. An FMEA has a column for "Recommended Actions" for each failure. However, that column indicates actions to prevent the occurrence of the failure, not to correct the failure if it should occur.
- FMEAs identify ultimate effects. FMEAs were constructed for the purpose of identifying potential problems so that actions could be taken to reduce the likelihood of their occurrence. Creating troubleshooting trees is a possible secondary use for the information -- it is preferable to avoid a problem than to provide tools for solving it when it occurs. This primary purpose dictates the type of information that is presented in the FMEA. In particular, the things that FMEAs call "effects" are often ultimate effects, not intermediate effects. Identifying ultimate effects makes it easier to assess the severity of the failure mode, but does not provide the causal connections for creating troubleshooting trees.
- FMEAs do not provide clues to which failure occurred.

Many of these shortcomings could be overcome with changes or additions to the FMEA process. But the first reason against using FMEAs -- that they provide sparse coverage of the causes behind any given problem -- is fundamental to the idea of analyzing the failure modes of a process. For that reason primarily the conclusion was reached that FMEAs are not suitable sources of information for creating troubleshooting trees.

### **5.3. A procedure for collecting troubleshooting information**

In this section an attempt is made to present a procedure for collecting the information necessary to build a troubleshooting tree. The procedure that was followed for this research is described first, then recommendations are offered for improving it.

In addition to storing the cause and effect information necessary to build a troubleshooting tree, the ideal information gathering process would:

- Be impervious to changes in wording. For example, if "taping problems" is a cause of one problem, and "poor taping" is an effect of another, there should be a way to get beyond the syntactic differences and recognize that these two phrases represent the same thing.
- Keep the levels of information consistent. That is, it should ensure that for each problem, its effects are somehow "one step away." It is desired that a problem will trace back to an upstream cause, which traces back to an upstream cause, which eventually traces to a root cause. Simply listing all root causes for each high-level problem will create a list structure, rather than a tree structure. The tree is more easily searched.
- It would be nice if the mechanism used for collecting the information could also be used to demonstrate the idea of a troubleshooting tree before the tree is fully constructed. One should be able to explain to another person how the application would work, then trace through the causes for one of the problems for which information has already been collected. This does not mean that the procedure has to be automated, or that it must have an interface resembling that of the final application.



The procedure that was followed for this research project is described in the steps below.

Step 1. Become familiar with the line, its stations, the product, and the terminology.

Step 2. Select a problem for which to build the first tree. Speak with those familiar with the problem, asking questions to understand what that problem means and some general things likely to have caused it. The idea of this step is to become familiar enough with the problem that it would be recognizable even if it were worded differently.

Step 3. Review all of the potential problem analyses that have been done. Gain some familiarity with the kinds of problems that have already been documented and their effects. If any of the potential problem analyses mention the problem for which the tree is being built, record the causes and any other information that is provided.

Step 4. Select individuals to interview to gain more information about the problem. Process engineers should be interviewed for a line that is already in production, and manufacturing development engineers for a line that is in development. Auditors (PMOs) and product engineers can suggest causes of defects found in audit. Technicians also should be included.

Step 5: Conduct the first round of interviews. Use a form such as the one in Figure 8 to record the problem cause information. Begin by filling in the name of the root problem, for example, "ink under tape," in the symptom column. Ask the interview subject, "When ink under tape occurs, do you know what to do to fix it, or does the solution depend on why there is ink under the tape?" If the answer is that there is a solution, ask what it is and enter it in the "solution" column of the row with ink under tape in the "symptom" column. If the answer is that the solution depends on the reason why there was ink under the tape, ask what things could have been the immediate cause of ink

under tape.<sup>10</sup> For each reply, make a copy of the original row with "ink under tape" in the "symptom" column and fill in each of the reasons on a separate row under the "possible cause" column. In either case, whether there is a solution or there are causes, check off K, M, or T depending on which line(s) the cause might apply to, and enter the name of the person who provided this cause under "source."

Symptom	Possible Cause	How to tell	Solution	Source	K	M	T

**Figure 8:** Troubleshooting tree data form

Suppose the interview subject said there were three possible reasons for ink under tape: poor encapsulant coverage, a wet pen, or a damaged pen body. The form would now resemble that in Figure 9.

Symptom	Possible Cause	How to tell	Solution	Source	K	M	T
ink under tape	poor encapsulant coverage			cmb	x	x	x
ink under tape	wet pens			cmb		x	
ink under tape	pen body damaged			cmb	x	x	x

**Figure 9:** Initial entries in a troubleshooting data form

Now take each possible cause and examine the form to see whether it is already listed somewhere in the "symptom" column. If it is not, write the cause on a new line under the

---

<sup>10</sup> It may take the subject a while to grasp what is meant by "immediate" cause. The idea here is to get at the closest previous thing that would have led to the problem. Thus "Someone at the ink supplier's plant had a cold the day this batch of ink was made" is not an immediate cause, but "Ink contaminated" might be.

"symptom" column. The form would now look like Figure 10. For the new symptoms, go back and ask the subject the question "Does it have a solution, or does it depend on why," and continue generating causes of the causes. The first portion of the data is complete when every possible cause is listed on its own line as a symptom, and every line has either a possible cause or a solution filled in. However, the data does not have to be complete to be useful.

Symptom	Possible Cause	How to tell	Solution	Source	K	M	T
ink under tape	poor encapsulant coverage			cmb	x	x	x
ink under tape	wet pen			cmb		x	
ink under tape	pen body damaged			cmb	x	x	x
poor encapsulant coverage							
wet pen							
pen body damaged							

**Figure 10:** Continuing the troubleshooting data form

The second step of the information gathering will involve filling in the "how to tell" column. For this research, the author found it preferable to interview many different subjects first, gathering possible causes and solutions where available, and then going back to fill in the "how to tell" column. Each new interview subject was asked for possible causes or solutions for all of the symptoms on the list, so that the causes that earlier people had offered were reviewed with each new subject. In this way, new causes for the root problem, ink under tape, could be added while the chains of reasoning that other subjects had begun were filled out. Especially in talking to the manufacturing development engineers, it turned out that each person was a specialist in a few areas, and so could usually fill in one branch of the causal tree completely (such as reasons for a pen being wet), but could give much less information on other branches of the tree. However, when they looked at the causes that others had suggested, they could often add

a new cause for a few of the decaying symptoms, things that someone with their expertise would know but no one else would have thought of.

Step 6. Save several hours between interviews to reorganize the data. It may be helpful to alphabetize the symptoms daily as the data is growing. The data for ink under tape was created as a table in Excel, which provides a function to sort rows alphabetically. This was helpful when deciding whether a cause was new, and so should be added as a new symptom, or whether it had been mentioned before and already existed in the list. But even alphabetizing got only part of the way toward answering that question. The real difficulty, and one that was encountered frequently, was to tell if a cause already existed when two people phrased it differently. For instance, should acronyms like "OP" or full words like "orifice plate" be used? Questions like this emphasize the importance of becoming familiar with the terminology before starting to collect the information. In some cases the sources had to be consulted again to determine whether one of their phrases and the phrase another source had suggested meant the same thing.

Step 7. Gather discriminating information, filling in the "how to tell" column in the form. This may be easier than collecting the causes, but it may also occasionally require restructuring parts of the causal trees. Return to each of the interview subjects, showing them the list, and pointing out cases where there were multiple causes for one symptom. For each such case, ask how they would tell which of the causes had occurred. For some cases a subject might answer. "Well, if this is the problem, then there will be a dark blot in the center. You can see it under a microscope." In other cases they might only be able to say, "I do not know how to tell, but I do know that this cause is much more likely than those other ones." Both kinds of information are useful and should be recorded in the "how to tell" column corresponding to the possible cause. That is, the "how to tell" entry is how to tell if the "possible cause" has occurred, not how to tell if the symptom has occurred.

Figure 11 contains a fragment of the troubleshooting information that was collected for this project. The data should provide a sense of the level of detail of the information that was collected. It also contains some examples of the wording difficulty alluded to above.

Symptom	Possible Cause	How to tell	Solution	Source	K	M	T
encapsulant dispense machine miscalibrated			recalibrate	pr	x	x	x
misplaced encapsulant bead	bad incoming materials: THA has bent or lifted beams		change to new THAs	pr's FMEA		x	
misplaced encapsulant bead	bubbles in encapsulant	examine under microscope -- compare to picture		pr's FMEA	x	x	x
misplaced encapsulant bead	contaminants in encapsulant	examine under microscope -- compare to picture		pr's FMEA	x	x	x
misplaced encapsulant bead	encapsulant dispense machine miscalibrated	encapsulant bead is the right size, but in the wrong place		pr's FMEA	x	x	x
tape application force too low	pneumatic crosstalk	other pressure regulators adjusted recently		tm		x	
tape application force too low	tape applicator misaligned with respect to pen	tape consistently adheres well on one side of the pen		tm		x	
tape application force too low	tape applicator pressure cylinder seals damaged	place pen with a load cell in fixture to measure force on pen. if pen sees different force than gauge says...		tm		x	
tape application force too low	tape applicator pressure set too low in cylinder	check setting		tm		x	
tape applicator force too high	tape applicator pressure too high			tm		x	
tape applicator misaligned with respect to pen	pen cocked in taper fixture			tm		x	
tape applicator misaligned with respect to pen	tape applicator damaged			tm		x	
tape applicator pressure cylinder seals damaged			replace cylinder	tm		x	
tape applicator pressure set too low in cylinder			reset pressure to spec	tm		x	

Figure 11: Troubleshooting tree data with clues

For the purposes of this project, information was gathered via interviews. All but one of the interviews were one-on-one, although interviews of a group might also have worked well. Handing people forms after explaining the procedure was not thought to be effective since much of the difficulty in collecting the data involved resolving wording differences and ensuring the right level of detail. These things were best accomplished by having a central person coordinating the data gathering.

It might have been better to collect the information in a database rather than in a spreadsheet. Which method is preferable depends on the chosen representation for the final troubleshooting tool.

One final suggestion for enhancing the data collection procedure would be to add a column for keywords relating to the symptom. This might have made it easier to find out if essentially the same cause was already entered, but just worded differently. A wildcard search might serve the same purpose.

## **6. Results and Conclusions**

This chapter presents the results of the research project. The current state of the project within HP is described, and the costs and benefits of creating troubleshooting trees are assessed. Based on the what has been learned from this research, recommendations for follow-on work are offered. The thesis concludes with a restatement of the main points.

### **6.1. Follow-on work by others**

The work described in this thesis has been continued by others within HP. In this section the additional work is summarized. Where it differed from the recommendations presented in this thesis, those differences are discussed and reasons for the differences considered. Reactions of the users to the current implementation are presented.

#### **6.2.1 Description of the tool**

The work carried on by HP also represented information as cause and effect connections, though the format chosen to present these troubleshooting trees to the user was changed. Section 3.2.2 describes representing troubleshooting trees in a database with a forms interface. That is the presentation that was chosen for the extended work carried out by HP.

A prototype system for storing, viewing, and editing troubleshooting trees was implemented in Visual Basic on a PC. Users may choose to "pick" (run through), "prune" (edit), or "plant" (create) a troubleshooting tree. The latter two functions are password protected so that only authorized users may change the trees.

When the user chooses to pick a tree, a list of problems for which troubleshooting trees are available is presented. The user selects a problem, and it is displayed as the start of the tree in a window containing the path traversed through the tree so far. The causes of

the problem are retrieved from a database and listed in a window of causes. The user may pursue any of the causes by double-clicking, in which case the selected cause is added to the path window, and its causes replace the previous contents of the causes window. At any point while a cause is selected, the user may elect to investigate that cause further before deciding whether or not to follow its path through the problem hierarchy. Three types of investigation are possible. The user may request to see a photo (a bitmap file) of how the problem would appear, a list of auxiliary symptoms that often occur in tandem with this cause, or the results of a query into the actual production database. This last function executes a canned query against the production database to determine things that may be relevant in ascertaining whether this cause has occurred. Queries that are thought to be likely include such questions as "How many times has this cause occurred in the past day?" or "When was this batch of raw materials last changed?" These three means of investigation serve the same purpose as the "clues" discussed earlier in this thesis.

## **6.2.2 Current users**

A training course was held on the use of the troubleshooting tree software, which was attended by a dozen people. An equal number of people have been trained informally. Between five and ten troubleshooting trees are currently being built.

The two largest troubleshooting tree implementations are for the M line and another new production line, "F." These efforts are being led by the manufacturing development engineers. For both lines' implementations, the development of the trees proceeds in the same way. Engineers, technicians, and operators meet to discuss a problem and identify possible causes. The engineers take this information and encode it into troubleshooting trees. The output of the troubleshooting trees (the tree structure represented as an indented list) will be posted by the machines on the F line. Operators will consult these lists to identify root causes and solutions to production problems they encounter. For



more complex problems the operators will consult an on-line version of the troubleshooting trees that will run on a PC by the line. The operator will traverse the troubleshooting tree, as described above, making use of the clues that were not present on the paper lists, to identify the root cause. Instead of describing a solution, the root cause within the tree will direct the user to the page of a specification document that describes the fix, which will also be available on-line. In this way the solution information will not have to be duplicated.

For the M line there will be a troubleshooting tree for every out-of-control event, except those for which there is only one root cause. Use of these troubleshooting trees will be included in the process control training for workers on the line. The goal is that the operators themselves should know how to return the process to control, and troubleshooting trees will be one means of achieving this goal.

In addition to the M and F lines, there are also several individuals who are building troubleshooting trees to serve as personal references. They are using the tool to organize their own knowledge about specific product details such as ink corrosion problems. One feature they find particularly useful is the ability to include scanned images within the trees.

### **6.2.3 Software implementation difficulties**

These early users have identified several difficulties in trying to build troubleshooting trees. Some problems had to do with the software tool, whereas others were conceptual. On the software side, they found the user interface for the tree-builder cumbersome to use. A graphical interface, with trees represented by nodes and links, would have been more intuitive than typing in each cause, and then its causes, and never seeing the whole tree. Early users identified a few bugs in the software, which were fixed.

The conceptual problems may be more difficult to solve. An engineer who was attempting to build a troubleshooting tree met with others to identify possible causes of a particular problem. The result of the meeting was a list of causes, rather than a tree structure. In addition, it was difficult to identify the clues that would help end-users determine which path through the causes to follow. It is hoped that the information gathering procedures included as section 5.3 of this thesis will provide assistance with this situation.

An additional conceptual problem dealt with the level of information to include in the tree. Tree-builders wanted to know where to start their trees. At what point should they end the tree and give a root cause? These questions arose because there are as yet no end-users for the troubleshooting trees, so the engineers are guessing at the level of information that will be helpful. Two guidelines offered in this thesis address these questions. First, start trees with observable, named problems. Second, end the tree when the choice of solution is unambiguous.

One problem that was foreseen did not occur in practice. It was predicted that users would encounter problems with wording, and create duplicate causes with slightly different names, thus replicating structures rather than reusing them. "Taping problems" and "poor taping" is one example of a wording problem that was encountered during this work. In practice, wording difficulties were not mentioned as a problem. Two features led to this. First, the troubleshooting tree software presented the tree-builder with a "pick list"<sup>11</sup> from which to choose causes. All of the causes that had been entered previously were presented as possible causes to be included for the branch of the tree currently being built. Pick lists were found to be so useful that they helped to define the size of the

---

<sup>11</sup> A pick list is a list of phrases that have already been used. The user either selects an item from the list, or types a new one. Pick lists are commonly used to promote reuse of existing items and reduce spelling mistakes.

trees. Since long pick lists are unwieldy, for one implementation it was decided to build more trees, thus leading to shorter pick lists. (A pick list included only those causes that had been entered for the current tree.) The second reason that wording problems have not occurred is that no sharing of branches among trees is yet occurring.

## **6.2.4 Management reactions**

Management has been quite supportive of the troubleshooting tree effort. The underlying concept of deeper and deeper causes of problems is both easy to understand and simple to communicate. Furthermore, the viewing interface to the troubleshooting trees is straightforward and intuitive to use, which was not the case with the interface for users creating the trees.

Two possible administrative problems have been identified. First, who will support the software for creating troubleshooting trees? Fortunately, the central Computer Integrated Manufacturing group has volunteered to support the application if its use continues to spread. Second, a need is seen for "data custodians." Who will do backups, maintain data security, and insure that the information in the trees is correct? The data will be "owned" by the maintenance technicians for each line. The system-level questions about backups and security will depend on whether the current implementation is continued, or whether a new implementation platform is pursued.

## **6.2 Cost/benefit analysis**

The cost of a troubleshooting tree can be divided into fixed (non-recurring) costs and variable costs. Fixed costs include nine months of engineering, which, at an estimated \$150,000/person-year comes to \$112,500. Instead of estimating variable costs per tree, since the same [set of] problem[s] could be represented as many trees or as a single one, cost is estimated for building however many trees are necessary to represent potential problems on a single module of a pen production line. It requires a conservative three

months of an engineer's time to collect the cause and effect knowledge, encode it, and enter the clues for one module of a production line. Add to this another month, total, from the technicians, operators, and other engineers with whom the tree builder consults, for a total of four months per module. Note that this investment certainly won't provide a tree containing every possible cause of problems within the module, but it should cover the 80% that are most likely to occur, which is sufficient to make the trees useful. Additional causes would be added as they are encountered in production. Total variable costs are therefore \$37,500 for the tree builder's time plus \$9,028<sup>12</sup> for the others consulted, or \$46,528 per module of the production line.

The returns on a troubleshooting tree come from a number of sources. First, there are the additional profits to be gained by increasing the uptime of a line producing a product for which demand outstrips supply. Second, there are savings in labor from both solving problems quicker and using less expensive workers to do so. Third, there is the possibility of deferring capital expenditures due to the increased output of production lines that have less downtime. Dollar values are estimated for the first two benefits, while the third is left as an additional benefit to be considered but not quantified.

Increased line uptime leads directly to higher profits, since all goods produced can be sold. If the profit on each pen is \$10 and the throughput of a line is 1,000 pens/hour<sup>13</sup>, then for each 1% that uptime is increased, profits would increase by  $1\% * (1,000 \text{ pens/hour}) * (24 \text{ hours/day}) * (365 \text{ days/year}) * (\$10 / \text{pen}) = \$876,000 / \text{year}$ . If troubleshooting trees could increase the uptime of a line from 90% to 95%, they would save \$4.38 million annually.

---

<sup>12</sup> Assuming engineers cost \$150,000/year, technicians \$100,000/year, and operators \$75,000/year, one month of time split equally among these three classes of workers costs  $1/3 * (\$150,000 + \$100,000 + \$75,000) / 12$  or \$9,028.

<sup>13</sup> Not the real values

The direct labor costs of solving a production problem are  $i * \$i + f * \$f$  where:

$i$  is the time required to identify the cause of the problem;

$\$i$  is the cost of an hour of time from the resource that identifies the cause of the problem;

$f$  is the time required to fix the problem; and

$\$f$  is the cost of an hour of time from the resource that fixes the problem.

Troubleshooting trees reduce the first two variables. First,  $i$ , the time required to identify the cause of the problem is shortened because many of the possible root causes will be included in the troubleshooting trees and therefore more quickly found. Second,  $\$i$ , the labor cost of the person identifying the cause is reduced because the troubleshooting trees allow the operators themselves to identify root causes of many problems, even if they have to call in a technician to implement the fix.

How much savings might the reduced labor costs account for in a year? If downtime due to production problems is estimated at 5% of available time, the maximum that could be saved is 5% of  $(24 \text{ hours/day}) * (365 \text{ days/year})$  or 438 hours. Perhaps 80% of this downtime is spent tracking down the root cause, with the remaining 20% spent implementing the solution. If the use of troubleshooting trees reduced the average time to identify the root cause of a problem by half, the time saved would be 1/2 of  $(80% * 438 \text{ hours})$  or 175 hours. If this troubleshooting time is divided equally between operators, technicians, and engineers, the saved time is worth \$9,479.<sup>14</sup> But the 175 hours that are still required to identify the causes of problems are spent primarily by lower paid employees. Instead of

---

<sup>14</sup>  $175 \text{ hours} * \frac{1}{3} * \frac{(\$150,000 / \text{year} + \$100,000 / \text{year} + \$75,000 / \text{year})}{2,000 \text{ hours / year}} = \$9,479$

being divided equally among engineers, technicians, and operators, problem solving may be performed by operators 80% of the time, with engineers and technicians each assisting in 10% of the difficult cases. This reduces the average cost of a problem-solving worker from \$54.17/hour<sup>15</sup> to \$42.50/hour<sup>16</sup>. For the 175 hours still spent identifying the root causes of production problems, the labor involved would cost 175 \* (\$54.17 - \$42.5) or \$2,042 less. Total savings from reducing problem-solving labor costs are \$9,479 + \$2,042 or \$11,521 per module per year.

The up-front investment of \$112,500 buys the opportunity to spend additional increments of \$46,528 per module in order to save \$887,521 per module per year, assuming only a 1% increase in uptime.<sup>17</sup> When additional factors are considered, the savings become even greater. Troubleshooting trees can be shared across lines, leading to quicker amortization of the costs of building the trees. The increased capacity that results from less line downtime may defer capital expenditures. And troubleshooting trees have a learning-curve effect: their value grows over time as the new causes that are identified are incorporated into the trees.

### 6.3. Recommendations

It is clear from the current state of the project within HP that there is a need for troubleshooting tools to be used on the production line. How well do the troubleshooting trees presented in this thesis meet that need? Engineers, the people who build the trees, find them to be appropriate for representing their knowledge. Despite early feedback on the drawbacks of the tree builder's user interface, the use of the tool is growing. It is time

---

<sup>15</sup> The average hourly rate of \$150,000/year engineers, \$100,000/year technicians, and \$75,000/year operators.

<sup>16</sup> 80% of the hourly rate for a \$75,000/year operator, plus 10% each of the hourly rate for a \$100,000/year technician and a \$150,000/year engineer.

<sup>17</sup> Using the disguised figures for throughput and profits per pen.

to involve the end users -- the operators on the production lines -- in the refinement of the end user interface. The portions of trees that now exist should be provided to a sample of operators, and their comments on the ease of navigating through the trees collected. Since there will be many more end users than tree builders, the simplicity of the end user interface is crucial. This end user interface should be the focus of the next effort on improving the troubleshooting tree software. It will be the key to the future success of the project.

It was never envisioned that troubleshooting trees would be built by engineers and then given, complete with all possible causes, to operators to use. The ability for operators or other users of the trees to add new causes as they are encountered in production will enhance the usefulness of the trees. If a problem is encountered that is not in the troubleshooting trees, it will require a good deal of time to identify its root cause. Once that root cause is found, if it is added to the tree, the large amount of time will not have to be invested the next time the problem is seen. By adding new causes once the trees are already in use, the trees themselves will serve as a repository for the knowledge the people associated with the production line have acquired. The trees will learn, too. A mechanism must be provided to allow end users to add new information to the trees. While there may be need for security, allowing only authorized people to change the structure of the trees, at the minimum a facility for annotating the trees with end user comments should be provided. These comments should be visible to other users of the trees as soon as they are entered. The owner of the tree, the person with the security to change the causal structures, should incorporate these notes into the trees themselves frequently; perhaps every week. When end users see that their additions to the trees are valued and have been incorporated, they will be more likely to continue adding to the trees. Without the operators' support, the trees will be of less and less use as their contents diverge from the reality of the production environment.

Following the previous recommendation, responsibility for extending the tool should be assigned. One individual could be assigned responsibility for periodically incorporating any new information that has been added into the tree structure. Centralization is needed to form the connections between the new information and the old, to scan in pictures or otherwise incorporate visual clues, and to investigate whether the new causes should be added to trees for other production lines as well.

The use of troubleshooting trees should be incorporated into training for operators, technicians, and engineers, as is planned for the M line. The availability of the tool must be publicized and its use encouraged.

Future work should investigate ways to share the information across production lines. Although a number of people are creating troubleshooting trees within ICD, none of the information in these trees will be shared with other trees. It will soon be time to address this issue. To that effect, the database paradigm discussed in section 3.2.2, which was the implementation chosen by HP, may be the best fit. One possibility for sharing branches of trees across lines would be to add to each cause in the database fields for the lines or products to which it applies. Then, if a user consults the tree for "ink under tape," only those causes of ink under tape that are possible for that particular production line would be brought up and displayed. But if one cause is common to several lines, that branch can be extended centrally, rather than existing separately in different trees. The primary difficulty here is one of coordination: who will control data, decide how to name causes, and ensure that branches are not replicated?

From this point, continuation of troubleshooting tree work within ICD can continue in one of two directions. Different groups, representing different products or even separate production lines could create stand-alone troubleshooting tools for problems encountered on their line(s), as is currently being done. Alternatively, each group could incorporate its trees into a centrally managed, globally accessible collection of trees. The



former option would be simpler to manage and execute, whereas the latter option would provide greater benefit by facilitating the sharing of problems and solution information across all production lines in the division. While combining all of the trees is a desirable goal for the long term, it does not appear to be a practical direction to pursue in the immediate future. Centralized coordination of the trees would involve procedures and a delay in incorporating information, and so deter people from extending the trees. The tool should be made as easily accessible as possible to encourage its use both in solving problems and in recording new causes of problems. The first alternative, that of allowing each group to build its own trees, is therefore recommended.

## **6.4. Conclusions**

Troubleshooting trees are appropriate representations for certain types of problems:

- There should be a name by which people refer to the problem.
- A significant portion of the time required to solve the problem today should be spent on tracking down the root cause, as opposed to implementing the solution.
- There must be people or other sources of information that can identify the majority of possible root causes of the problem.
- Most potential users of the troubleshooting tree should not already be able to identify the root causes of the problem.
- The users of the tool should be reasonably knowledgeable about the problems that may occur, and should be highly enough skilled that they can investigate problem causes with only guidance from the tool.

FMEAs have been considered as a vehicle for collecting troubleshooting information and rejected. That is not to say that they are not useful. FMEAs provide benefits by identifying potential problems early and highlighting actions to be taken to prevent their occurrence.

Even if a troubleshooting tree were not to be implemented in software, the process of capturing and structuring the causes and effects of problems has useful consequences. In ICD, once people became aware that someone was collecting the causes of one particular problem, they began relaying new problem causes soon after they had been solved so that the information could be recorded.

Troubleshooting trees are applicable to problems where there is a wealth of causal information, but specific causes occur infrequently enough that many are forgotten. For the trees to be useful, their users must be familiar with the problem domain, and comfortable examining potential causes themselves.

# Bibliography

- [1.] "Representing Structure and Behavior of Digital Hardware," Randall Davis and Howard Shrobe, *IEEE Computer*, October 1983, p. 75 -- 82
- [2.] "Diagnostic Reasoning Based on Structure and Behavior," Randall Davis, *Artificial Intelligence*, V. 24, 1984, p. 347 -- 410
- [3.] "Model-based Reasoning: Troubleshooting," Randall Davis and Walter Hamscher, chapter 8 in Exploring Artificial Intelligence, by Howard Shrobe *et al*
- [4.] Ford's Potential Failure Mode and Effects Analysis (FMEA) instruction manual (revised 9/88)
- [5.] "Risk Analysis and Management," M. Granger Morgan, *Scientific American*, July 1993, p. 32 - 41
- [6.] A First Look at Fault Trees, Alvin W. Drake
- [7.] An Introduction to Fault Trees, Alvin W. Drake, May 24 1984
- [8.] Introduction to Fault Tree Analysis, J. L. Recht
- [9.] "Introduction to Fault Tree Analysis," R. E. Barlows and H. E. Lambert in *Reliability and Fault Tree Analysis, Theoretical and Applied Aspects of System Reliability and Safety Assessment*, p. 7 - 35, SIAM 1975
- [10.] Fault Tree Handbook, W. E. Vesely, F. F. Goldberg, N. H. Roberts, D. F. Haasl, US Nuclear Regulatory Commission, NUREG - 0492, January 1981
- [11.] "ALF-A: A Knowledge Acquisition Tool for Troubleshooting of Laboratory Equipment," Henrik Eriksson and Per Larses, in *Journal of Chemical Information and Computer Sciences* Volume 32 Number 2, 1992, p. 139 - 144
- [12.] "Use Knowledge-Based-System Programming Toolkits To Improve Plant Troubleshooting," Prasad Ramanathan, Suresh Kannan, and James F. Davis, *Chemical Engineering Progress* Volume 89 Number 6, 1993, p. 75 - 84
- [13.] "Case-Based Reasoning: A Research Paradigm," Stephen Slade, *AI Magazine*, 1991, p. 42 -- 55
- [14.] "A Case-Based Reasoning Solution to the Problem of Redundant Resolutions of Nonconformances in Large-Scale Manufacturing," Stuart J. Brown and Lundy M. Lewis, p. 121 -- 133 in *Proceedings of the AAAI Conference on Innovative Applications of Artificial Intelligence 3*, AAAI Press, 1991, Reid G. Smith and Carlisle Scott eds.
- [15.] "Automated Assembly of the HP DeskJet 500C/DeskWriter C Color Print Cartridge," by Lee S. Mason and Mark C. Huth, August 1992 *Hewlett-Packard Journal*, p. 77 -- 81