A

Seminar report

On

Poka-Yoke

Submitted in partial fulfillment of the requirement for the award of degree
of Electronics

SUBMITTED TO:
www.studymafia.org

SUBMITTED BY:
www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Poka-Yoke**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Introduction

Poka-yoke is a quality assurance technique developed by Japanese manufacturing engineer Shigeo Shingo. The aim of poka-yoke is to eliminate defects in a product by preventing or correcting mistakes as early as possible. Poka-yoke has been used most frequently in manufacturing environments.

Hewlett Packard currently develops its Common Desktop Environment software to run in twelve locales or languages. Traditional testing of this localized software is technically difficult and time-consuming. By introducing poka-yoke (mistake-proofing) into our software process, we have been able to prevent literally hundreds of software localization defects from reaching our customers.

This paper describes the poka-yoke quality approach in general, as well as our particular use of the technique in our localization efforts. Poka-yoke is providing a simple, robust and painless way for us to detect defects early in our localization efforts.

What is Poka-yoke?

Poka-yoke is a Japanese term that means "mistake-proofing". A poka-yoke is any mechanism in a lean manufacturing process that helps an equipment operator avoid (*yokeru*) mistakes (*poka*).

Its purpose is to eliminate product defects by preventing, correcting, or drawing attention to human errors as they occur.[1] The concept was formalised, and the term adopted, by Shigeo Shingo as part of the Toyota Production System. It was originally described as *baka-yoke*, but as this means "fool-proofing" (or "idiot-proofing") the name was changed to the milder *poka-yoke*.

History

The term poka-yoke was applied by Shigeo Shingo in the 1960s to industrial processes designed to prevent human errors.Shingo redesigned a process in which factory workers, while assembling a small switch, would often forget to insert the required spring under one of the switch buttons.

In the redesigned process, the worker would perform the task in two steps, first preparing the two required springs and placing them in a placeholder, then inserting the springs from the placeholder into the switch. When a spring remained in the placeholder, the workers knew that they had forgotten to insert it and could correct the mistake effortlessly.

Shingo distinguished between the concepts of inevitable human mistakes and defects in the production. Defects occur when the mistakes are allowed to reach the customer. The aim of poka-yoke is to design the process so that mistakes can be detected and corrected immediately, eliminating defects at the source.

Categories of poka-yoke devices

Poka-yoke devices fall into two major categories: *prevention* and *detection*.

A *prevention* device engineers the process so that it is impossible to make a mistake at all. A classic example of a prevention device is the design of a 3.5 inch computer diskette.The diskette is carefully engineered to be slightly asymmetrical so that it will not fit into the disk drive in any orientation other than the correct one. Prevention devices remove the need to correct a mistake, since the user cannot make the mistake in the first place.

A *detection* device signals the user when a mistake has been made, so that the user can quickly correct the problem. The small dish used at the Yamada Electric plant was a detection device; it alerted the worker when a spring had been forgotten. Detection devices typically warn the user of a problem, but they do not enforce the correction.

We are surrounded every day by both detection and prevention poka-yoke devices, though we may not usually think of them as such. My microwave will not work if the door is open (a prevention device). My car beeps if I leave the key in the ignition (a detection device). At few years ago, some cars were designed not to start until the passengers had buckled their seat belts (a prevention device); but this mechanism was too intrusive and was replaced by a warning beep (a detection device).

Characteristics of good poka-yoke devices

Good poka-yoke devices, regardless of their implementation, share many common characteristics :

- they are simple and cheap. If they are too complicated or expensive, their use will not be cost-effective.
- they are part of the process, implementing what Shingo calls "100%" inspection.
- they are placed close to where the mistakes occur, providing quick feedback to the workers so that the mistakes can be corrected.

Judged by these criteria, the "small dish" solution to the missing-spring problem is an excellent poka-yoke device:

- It was simple.
- It was cheap, involving only the cost of a small dish.
- It provided immediate feedback about the quality of the work; corrections could be made on the spot.

Why is it important?

Poka-yoke helps people and processes work right the first time. Poka-yoke refers to techniques that make it impossible to make mistakes. These techniques can drive defects out of products and processes and substantially improve quality and reliability.

It can be thought of as an extension of FMEA. It can also be used to fine tune improvements and process designs from six-sigma Define - Measure - Analyze - Improve - Control (DMAIC) projects.

The use of simple poka-yoke ideas and methods in product and process design can eliminate both human and mechanical errors.

Poka-yoke does not need to be costly. For instance, Toyota has an average of 12 mistake-proofing devices at each workstation and a goal of implementing each mistake-proofing device for under $150.

When to use it?

Poka-yoke can be used wherever something can go wrong or an error can be made. It is a technique, a tool that can be applied to any type of process be it in manufacturing or the service industry. Errors are many types –

1. **Processing error**
   Process operation missed or not performed per the standard operating procedure.

2. **Setup error**
   Using the wrong tooling or setting machine adjustments incorrectly.

3. **Missing part**
   Not all parts included in the assembly, welding, or other processes.

4. **Improper part/item**
   Wrong part used in the process.

5. **Operations error**
   Carrying out an operation incorrectly; having the incorrect version of the specification.

6. **Measurement error**
   Errors in machine adjustment, test measurement or dimensions of a part coming in from a supplier.

How to use it?

Step by step process in applying poka-yoke:

1. Identify the operation or process - based on a pareto.

2. Analyze the 5-whys and understand the ways a process can fail.

3. Decide the right poka-yoke approach, such as using a
   shut out type (preventing an error being made), or an
   attention type (highlighting that an error has been made) poka-yoke
   take a more comprehensive approach instead of merely thinking of poka-yokes as limit switches, or automatic shutoffs

4. Determine whether a
   contact - use of shape, size or other physical attributes for detection,
   constant number - error triggered if a certain number of actions are not made
   sequence method - use of a checklist to ensure completing all process steps is appropriate

5. Trial the method and see if it works

6. Train the operator, review performance and measure success.

Qualities of a Good Poka Yoke

A good Poka Yoke must meet the following qualities:

- **Early:** A good poka yoke must be early in the process, so that it can provide quick feedback -- and help in detecting mistakes the moment they occur.
- **Precise:** It should be precise, so that it is easy to diagnose and identify what mistake occurred.
- **Simple:** The poka yoke should be simple -- to develop and maintain. This is quite important since one doesn't want to spend time and effort in maintaining poka yokes, and complex poka yokes will have a fairly high chance of becoming erroneous. Having a buggy poka yoke is worse than having no poka yoke at all.
- **Light:** The poka yoke needs to be unobtrusive and transparent. If a poka yoke itself becomes an overhead to the process, then it will drive the developers/users crazy, and they will find ingenious ways to avoid it all together. For instance, think about how a developer will feel if he/she has to run a 70 minute pre-commit script before each and every check-in!

Poka-yoke and Localization

Some background on message catalogs and localization

To create POSIX-compliant software that runs in multiple locales, developers store locale-specific strings in files called message catalogs. Rather than hard-code a text string into the application, a developer stores the text string in the message catalog and references it by its message set and message number. A message set and message number uniquely identify any message string in the catalog.

Localization is the process of creating a message catalog for a particular language. Hewlett-Packard currently localizes its Common Desktop Environment software for 11 locales: French, German, Italian, Korean, Spanish, Swedish, plus 2 Japanese locales and 3 Chinese locales.

Localization is typically done after the development of the software has stabilized, and it is typically done by people external to the core development organization. These people, called localizers, receive the application's message catalog from the development organization. They then translate each message string into its equivalent expression in the target language.

Localizing a software application is a difficult job. The localizers may be unfamiliar with the application. They may be located halfway around the world from the development organization. They may not even be familiar with programming. Usually, therefore, the localizer performs translations based almost exclusively on the contents of the message catalogs and the information provided in the localization documentation.

Testing localized software

Testing localized software poses a unique set of challenges. The localizers know what the translated messages say, but since they may never have seen the application run, they cannot know if their translation is correct. The development team knows what the translated messages are supposed to say, but since they are not familiar with the target languages, they cannot know if that is what the translated message actually says. Given the constraints of time and distance, it is difficult for localizers and developers to work together, especially when localization is being done in 11 languages.

The usual testing approaches do not offer much relief. Running the tests manually can become tedious when there are 11 foreign locales to test. Testers become fatigued running the same test in multiple locales.

The traditional way to test the message catalogs is as follows:

- the test team receives the translated message catalog from the localizer

- the test team installs the new message catalog and executes the test plan in the target locales
- obvious mistakes are referred back to the localizer and incorporated into a later release of the catalog.

This approach has several drawbacks. It is difficult to execute a test automatically in multiple locales. For one thing, image comparisons are not portable across locales, since by the definition of localization something should change on the screen when moving to a new locale. The alternative -- recording golden images in a dozen locales -- would be a maintenance nightmare.

Yet some sort of testing of localized software is necessary because there are many opportunities for a localizer to make mistakes when creating a message catalog. A localizer could

- specify an application menu incorrectly
- inadvertently delete a message string
- specify an invalid data format
- specify an invalid conversion format
- neglect to translate a message
- translate a phrase incorrectly due to lack of context

All of these mistakes have different causes and different effects on the software that is delivered to the customer. It is, however, possible to construct poka-yokes to counteract each of these mistakes. As an example, we will go into some depth about the poka-yoke we created to mistake-proof the localized application menus.

Of mice and menus

Many software users navigate through menus using only their mouse, clicking on the selections they want. But users can invoke menu actions without a mouse, by using menu mnemonics. Mnemonics are single characters (usually underlined) in a menu label. If the mnemonic is typed at the keyboard while the menu is displayed, the associated action is invoked, just as if the user had selected the action with a mouse. For instance, in the English locale menu shown in Figure 1, the selection that will close the application window has the label "Close" and the mnemonic "C".

Figure 1: Text Editor File menu in English and French locales

Application menus are prime candidates for translation into various languages. How else can users unfamiliar with English know what options are being offered? And since we are translating each menu label, we must also translate the mnemonic associated with each label. In the French locale menu shown in Figure 1, for instance, the selection that closes the window has the label "Fermer", and the associated mnemonic "F". In their respective message catalogs, the English and French menus appear as follows:

$set 11	$set 11
17 N	17 N
18 New	18 Nouveau
19 O	19 O
20 Open ...	20 Ouvrir ...
21 I	21 I
22 Include ...	22 Inclure ...
23 S	23 S
24 Save	24 Sauvegarder
25 A	25 a
26 Save As ...	26 Sauvegarder sous ...
27 P	27 p
28 Print ...	28 Imprimer ...
29 C	29 F
30 Close	30 Fermer

Table 1: English and French locale menus in message catalogs

Conclusion

Poka-yoke scripts like the ones described here can eliminate entire classes of errors. And once the scripts are in place they can run automatically without human intervention, raising an alarm only when a problem is discovered. The scripts we used provided quick feedback early in the process, detecting localization mistakes before the application ever reached the formal testing phase.

The poka-yoke approach proved very flexible, allowing us to validate aspects of the menus early in the development, even without the associated applications. However, it is useful to keep in mind that verifying the menus syntactically is not the same as testing the menus in the application. The poka-yoke scripts did not verify that the menus actually worked; that would require running the application.

The poka-yoke approach provided a simple and robust way for us to detect and correct localization mistakes that would have been difficult to detect through traditional system testing.

Reference

www.wikipedia.com
www.google.com
www.studymafia.org