LEAN SIX SIGMA

The Dirty Thirty Process for Six Sigma Software

by Jay Arthur

QIMacros®

Table of Contents

The Dirty Thirty Process for Six Sigma Software

Quantify the Costs	1
Understand the Pareto Pattern	2
Analyzing the Dirty Thirty	3
Analyzing Results	4
Common Problems	5
Conclusion	6
About Jay Arthur	6

The Dirty Thirty Process for Six Sigma Software

by Jay Arthur

While most software quality efforts focus on requirements, design, code and test, this method focuses on fine tuning delivered software. Yes, it would be better to prevent the kind of problems we see in software, but applications continue to be written by people using requirements and designs that can be flawed. Software is rarely released, it *escapes*.

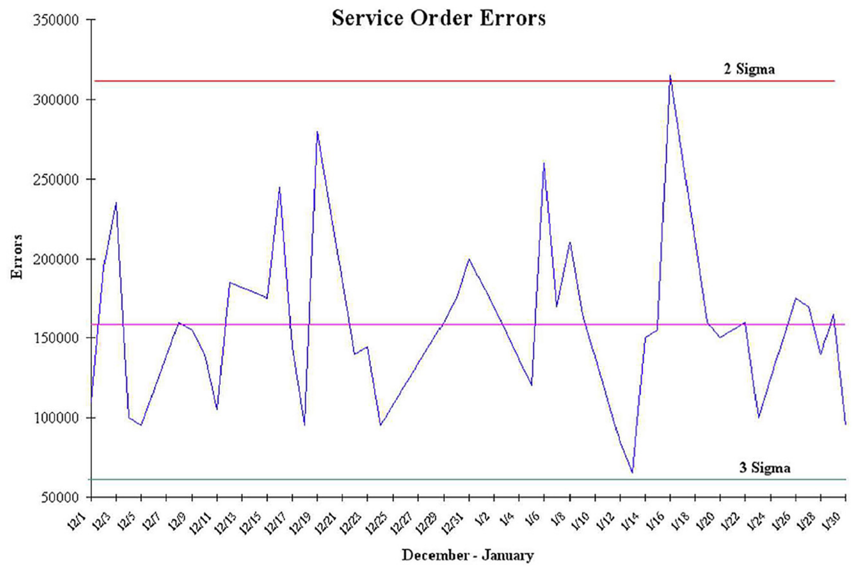
IT managers and application users often expect a new software project or enhancement release of an application to be flawless, and then are stunned by the additional staffing required to stem the tide of rejected transactions.

The secret is to:

1. Quantify the cost of correcting these rejected transactions
2. Understand the pareto pattern of rejected transactions
3. Analyze 30 rejected transactions one by one to determine the root cause
4. Revise the requirements and modify the system to prevent the problem

Quantify the Costs

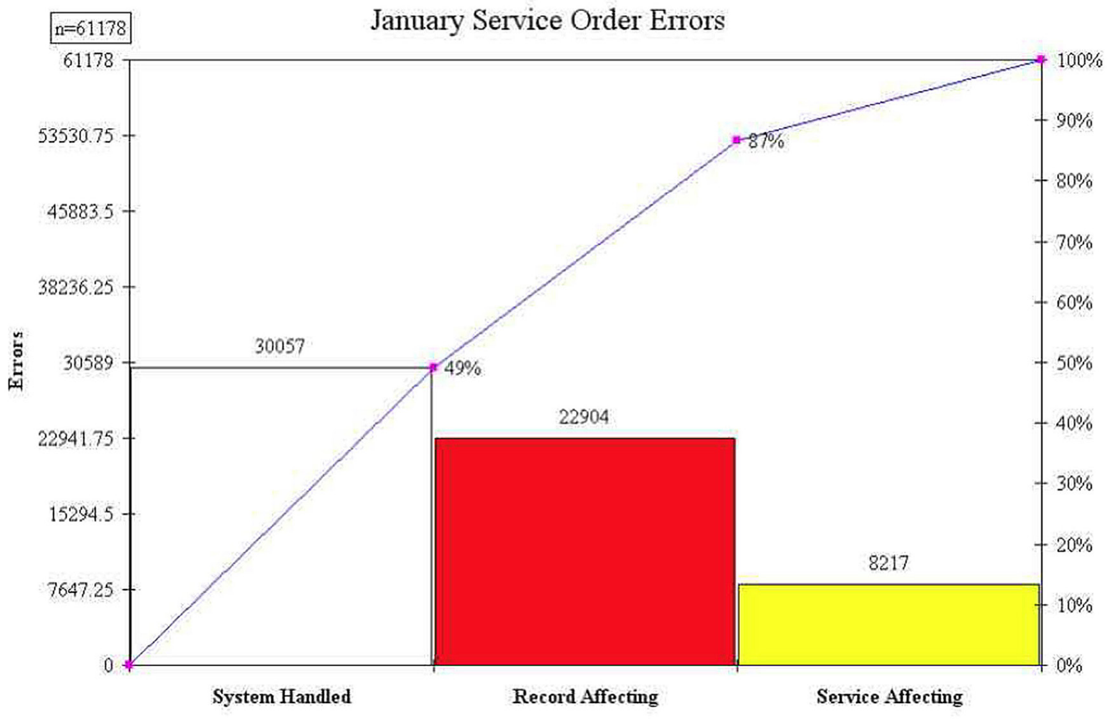
The first step in the "Dirty Thirty" process is to identify the number of rejected transactions and the associated costs. In working with one wireless company we found a 17 percent level of rejected service orders (170,000 parts per million):



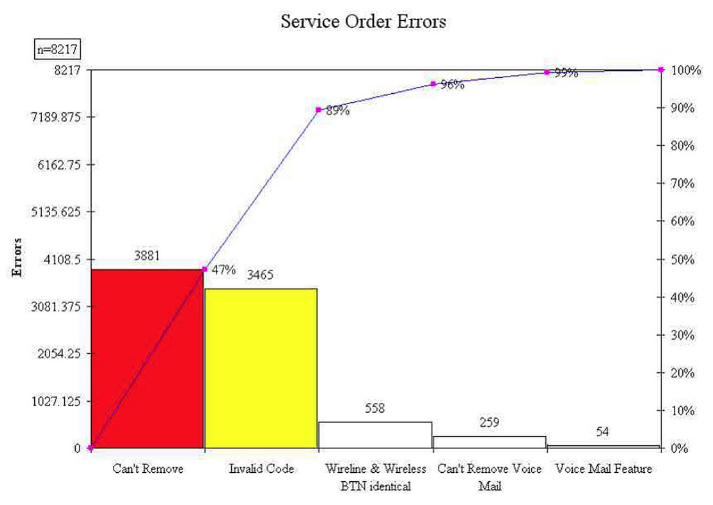
There were over 30,000 errors per month which, at an average cost of $12.50 to fix (correction group wage cost only), cost $375,000 per month. Over 50 temporary workers had been hired to deal with the backlog of unfixed errors. The objective was to cut this level of rejects in half by the end of the year.

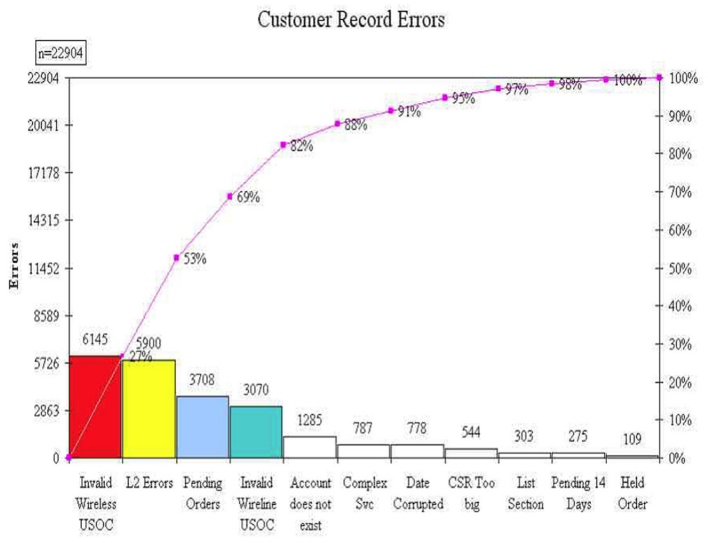
Understand the Pareto Pattern

All systems have routines to accept, modify, or reject incoming transaction data. These are assigned error codes and dumped into error buckets to await correction. In the service order system, the application handled much of the modification, but it still left significant quantities of defects to be corrected manually:



There were over 200 different error codes, but only six of them accounted for over 80 percent of the total rejected transactions. Two affected service directly; four affected the customer records:





It only took about three days to gather the data and isolate these transactions as the key ones to focus on.

Analyzing the Dirty Thirty

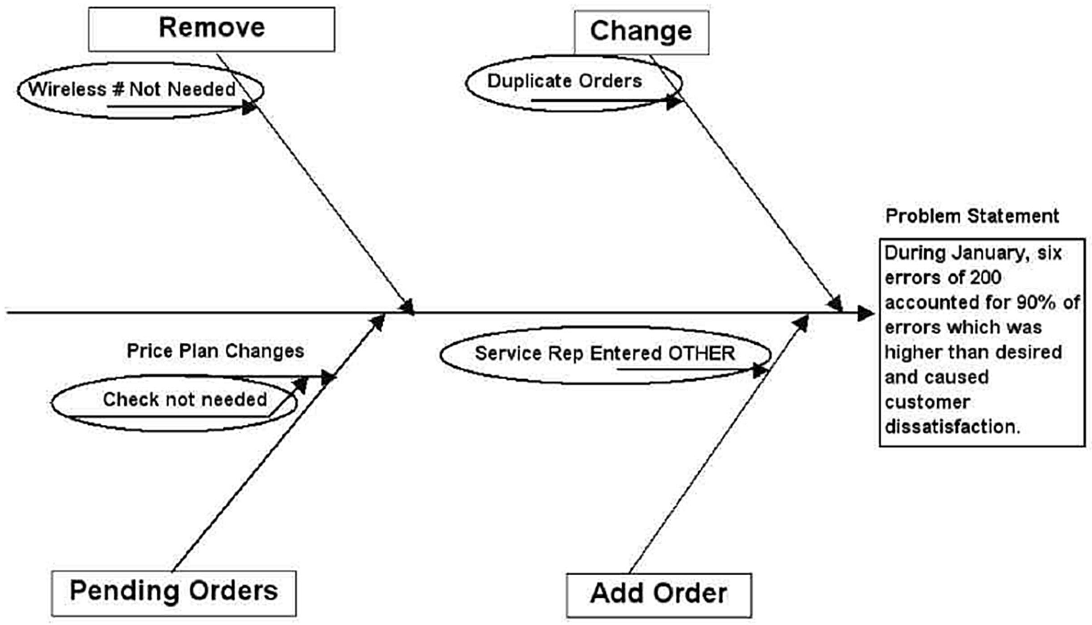
The next step was to convene root cause teams to investigate 30 rejects of each error type. It took a week or more to get the right people in the room to investigate each type of error. The right people included the IT systems analyst, error correction people, and service order entry. To attempt to do all six at one time with the same people would have been foolish.The errors required different subject matter experts and the root causes were too different. By restricting ourselves to just one error type per team, we were able to find the root causes in just one half-day meeting per team.

To prepare for the meeting we printed out 50-100 examples of each error. Then,

1. Using all of the on-line systems, we investigated the root cause of *each* rejected transaction. Again, we restricted ourselves to analyzing just one transaction at a time.

2. As the team agreed on the cause of the rejected transaction, I kept a stroke tally for each root cause. Gradually, as we looked at more and more transactions, a pattern would reveal itself. Sometimes it only took 25 transactions, sometimes it took 50, but a pattern would reveal itself clustered around one or more root causes. The great thing about evaluating transactions one at a time is that you verify your root causes as you go.

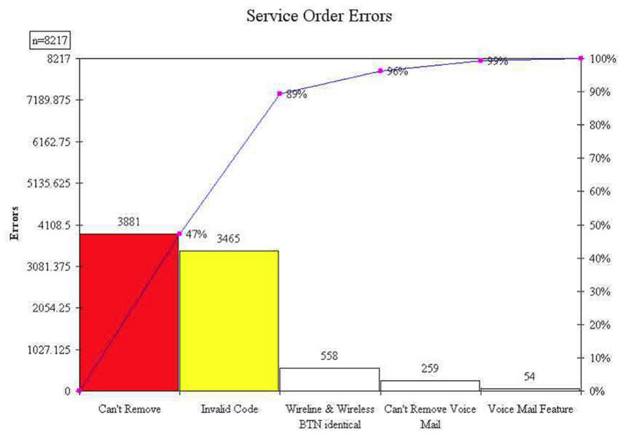
3. Once the team had identified the root causes, we would stop analyzing and spend an hour defining the new requirements. Most of the time, the requirements were too tight, sometimes too loose, and occasionally nonexistent. The systems analyst would then convey these to the programming staff for implementation.

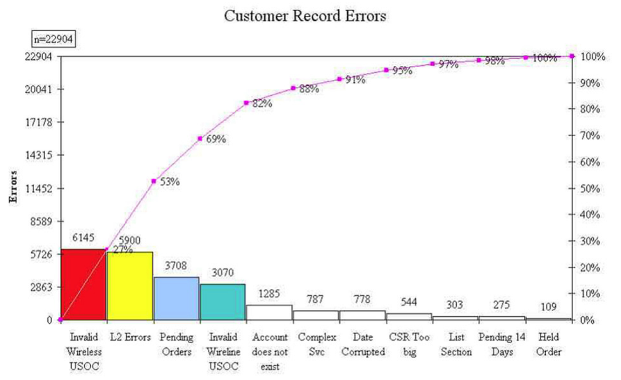

Remove
Wireless # Not Needed
Change
Duplicate Orders
Problem Statement
During January, six errors of 200 accounted for 90% of errors which was higher than desired and caused customer dissatisfaction.
Price Plan Changes
Check not needed
Service Rep Entered OTHER
Pending Orders
Add Order

Analyzing Results

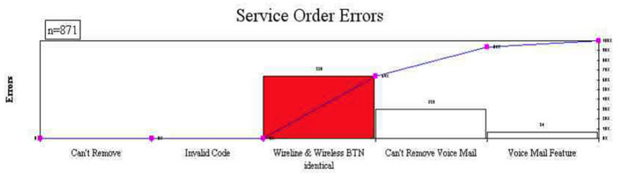
It took four months to implement the revisions, but it was worth it. By midyear the changes *completely eliminated the two top service-affecting errors, and three of the four record-affecting changes*. It cut total errors from 31,121 down to 7,167 per month — a 77 percent reduction in total errors. This reduction translated to $299,426/month in savings—over $3 million per year.

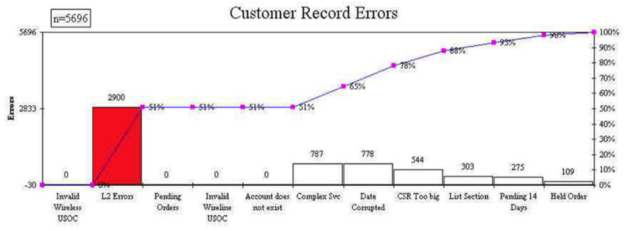
Before





After





Common Problems

The core elements of any application involve searching for and then adding, changing, or deleting data. Most applications assume a perfect world, where the data is only created or modified by the system. This is rarely the case. Most systems have a wealth of backdoors used to fix faulty data quickly. Upstream and downstream systems have their own backdoors to fix faulty transactions, so perfect data continues to be a mythological assumption that fosters faulty requirements and designs.

The requirements for adding, changing, and deleting data are often:

- Too loose
- Too tight
- Nonexistent

which leads to errors and rejected transactions that must be corrected manually by people hunched over computer terminals for eight hours a day.

Conclusion

Until you get to where you can prevent errors in requirements, design, code and test, every system release could benefit from a simple, yet rigorous approach to analyzing and eliminating post-implementation errors. The "Dirty Thirty" process is ideal because the data required to implement it is collected by most systems automatically Then all it takes is four-to-eight hours of analysis to identify the root cause of the error. Most of the time, the root cause will reside in the requirements.

One of the positive by-products of this approach is that the systems analysts learn first hand how their requirements and designs most often fail. This allows them to learn how to make their next set of requirements or designs more robust. It also gives the user a closer look at the intricacies of software and the complexities involved. And, if you aren't going to start using the Dirty Thirty process, what are you going to use to mistake proof your systems and releases?

Until software engineering finds ways to prevent all of the possible defects inherent in software development, the Dirty Thirty process will provide a simple way to tune up a system release and move it ever closer to Six Sigma performance.

About Jay Arthur

Jay Arthur, the KnowWare Man, teaches people how to eliminate delay, defects and deviation in one day using Excel and the Magnificent Seven Tools of Lean Six Sigma. Jay is the shortcut to results with Lean Six Sigma.

Jay is first and foremost a Money Belt; he knows how to use data to fix broken processes to save time, save money and save lives. Jay has 25 years of experience helping companies save millions of dollars.

Jay is a frequent speaker at Lean Six Sigma conferences and is the author of many popular Lean Six Sigma books published by McGraw Hill including **Lean Six Sigma Demystified** and **Lean Six Sigma for Hospitals**. He is also the developer of **QI Macros Software for Excel**.