

October 15, 2018

Report on Interviews with Key IPC Staff Members and Customers

# Summary

On September 30, on behalf of UTA, I accepted a commission from Rachel Wallace of IPC to do a DMAIC analysis of their software failure and customer response problems. Rachel is IPC’s software maintenance manager. There appear to be two problems:

1. An abnormally high failure rate for some of their software products.
2. Slow response when failures are reported to IPC. In some cases, it takes 4 months between the time the failure is reported and the time the fix is installed and working at the customer’s site.

To help with analysis of these problems, I have obtained the following information:

* This account of what various IPC individuals have to say about these issues
* An IPC organization chart, which can be used to show how the various IPC individuals relate to each other
* Copies of various letters and memos related to this activity
* An IPC Product Data spreadsheet showing key metrics that have been collected and computed over the past three years.
* Various other IPC data that may be of relevance

This report is the edited record of a series of interviews I conducted with IPC staff members and one of their customers over the past two weeks. My goal was to obtain some factual information in order to perform the analysis. I have also initiated a DMAIC plan by working closely with several key IPC managers. The next step of this process will enlist the aid of UTA graduate students to analyze the attached interview records as well as the other information described above. They will use this to update the DMAIC plan. The students’ plan will contain the following information:

* Identification of CTQs (measures that are critical to quality).
* Descriptions of the software maintenance (customer response) process (described via a swim lane chart and accompanying textual write-up), identifying areas of concern such as excess complexity, excessive delays, non-value-added tasks and other factors contributing to the delays reported by the customer.
* Analysis of the product data to see if that provides any insights.
* Root cause analysis, identifying the causes and, ultimately, the root causes of the two problems.
* Causal models, outlining all causes of each problem.
* Selection of three most important root causes and further analysis of each.
* Recommended actions for IPC to improve the situation.

I will send these reports to IPC for their analysis and action.

# Detailed Interview Reports follow, starting on the next page

Interview Report

Person Interviewed: Lars Johanson

Job Title: Information Technology Manager, ACME Corporation [an IPC customer]

Description of job: Responsible for providing computing and networking services to all ACME employees.

Failure Problems: We do not use all of IPC’s products, but we do use a lot of them. Although many of IPC’s products have what I would classify as a normal level of failures, certain products we started using in the past several years have had abnormally high failure rates. Perhaps the worst is the Zillow product, but we also have high failure rates with several others, notably Zorro, Halo, Casper and Justify. These are vital products for our business.

Failure report processing: IPC provides a lot of our important software and when we have a failure attributed to that software I contact IPC’s customer representative, Jeff Arterburn, and give him the particulars. My organization also tests out any updates to the software after Jeff installs it.

How Long it Takes: For all product failures I notify Jeff within one working day and it takes us no more than 1 day to test any updates that we receive from IPC. When the problem is urgent or critical, which should never happen but actually does happen now and then, IPC sends us a software developer to attend to the problem. Usually they work with our staff to devise a work-around solution until they can hunt down the actual source of the problem and fix it. Unfortunately, fixing the problems, even the important ones, takes longer than we think it should: usually at least 3 and sometimes as much as 5 months, between the time we notify Jeff of the problem and the time we have an installed update that fixes the problem. If it’s a low priority (minor) problem, it may take even longer.

Their Assessment: The high failure rates seem to be concentrated in only a few products, so perhaps something about the way IPC develops these products is a problem. As to the delays in getting problems fixed, their approach does not satisfy reasonable customer needs. They seem to have very good technical people, but something is clearly wrong.

Interview Report

Person Interviewed: Jeff Arterburn

Job Title: Customer Representative

Description of job: IPC’s representative to some of its major customers, including ACME, Delta, Zephyr and Zeta Corporation.

Failure Problems: Certain products in the past several years have had an abnormally high number of failures. I don’t know what has happened at IPC to cause this, but the problem seems to be concentrated in only about one third of the products.

Failure Report Processing: When a customer contacts me with a software failure I collect the relevant information and complete a problem report (on line). That report goes to our maintenance clerk (George Wilson). If the problem is severe enough to halt the customer’s business operations, we classify it as critical. In such a case, I talk directly with the maintenance manager (Rachel Wallace) and she sends a software developer to the customer’s site immediately. However that doesn’t happen very often. Usually the problem is something the customer can work around for a while until the IPC home office finds a more permanent fix. I can go on line to see the status of the problem report as it makes its way through the IPC system and I give the customer weekly status reports (or I can give an immediate report upon request). When the permanent repair is delivered to me I install it at the customer’s site and they run tests to make sure it is OK. It usually works fine at that point, but if it doesn’t I have to send it back to the home office so they can see what went wrong.

How Long it Takes: I turn in the problem report within one day. However it often takes 3-5 months for the permanent repair to get back to me. From watching the status reports, I see the following stages:

* **Waiting to be assigned**. This is the initial stage once I send it in, and the length of time in that stage varies with the severity of the problem. For a typical problem it often takes several weeks. For a minor problem it may take over a month. But for a critical problem we put it at the top priority and it only takes a few days for them to start working on it.
* **Assigned** and waiting to be fixed. This means a programmer has been assigned to fix the problem. It usually takes 3-4 weeks but sometimes takes several months before the problem has been fixed. Occasionally (about one time in four) the problem will move back to the “**waiting to be assigned**” stage after having been in the **“assigned”** stage for a while. I don’t know why this happens, but in cases like this it will typically be another month before it gets back to the **“assigned”** stage. I sometimes wonder if the programmer rejected it because the fix was too hard, but that’s probably not what’s really happening. It is frustrating to report this to the customer because it looks like we’re going backwards!
* **Fixed** and waiting for regression test. This means the programmer has fixed the problem, or at least they believe they have fixed the problem, and it has passed the unit and system tests. The wait for regression test sometimes takes a month, but is usually a about half that. (Regression test is where they perform a comprehensive test on the entire product to make sure the fix for this problem did not cause a problem somewhere else. Regression test is expensive to perform and takes several days, so they often hold up regression test until several fixes for different problems have been completed.)
* **Tested** and waiting for final approval. This means it has passed through the regression test and is waiting for a final update to be prepared and final approval. This can take anywhere from a day to a month. It’s very frustrating when it takes so long.
* **Waiting for shipment**. This is the last step before the repair is sent to me. This step is usually pretty fast – one or two days.

Their Assessment: The high failures only seem to happen with certain products, so someone ought to look into whether there’s a common factor affecting all of those products. As to the delays in fixing problems, from what I can tell by looking at the status, there are several bottlenecks in the IPC system. Except for final shipment, each stage in the process seems to take longer than it should, but I don’t know why. I don’t know enough about the specific details to know exactly what causes these steps to take so long.

Interview Report

Person Interviewed: George Wilson

Job Title: Maintenance Clerk

Description of job: Review incoming problem reports, prioritize them, combine similar reports that probably require the same fix, and ship updates back to the customer rep.

Failure Problems: Failure rates are unusually high for certain of our products. I don’t know of any common factor that would explain this, but Sharleen, the maintenance programming manager, can provide some failure rate data that show more detailed measurements of all our products.

Failure Report Processing: Every morning I look at the incoming problem reports and process them as follows:

1. Once they come to me they are given a unique tracking number and are in the “**waiting to be assigned**” stage. I give each problem a priority based on its severity, on a scale of: minor, normal, critical. (Critical is where some customer’s operation is disrupted or sometimes when many different customers have complained about the same failure.) The customer representative usually has assigned a severity code but sometimes I disagree with them. In that case I ask the maintenance manager (Rachel Wallace) what severity to assign. This sometimes takes a week or more because Rachel and the customer rep have to find a time when they can discuss it and their schedules don’t always match.
2. If I find any critical problems, Rachel contacts Sharleen, the maintenance programming manager, and arranges to send someone to the customer’s site. But that only happens about once a year. For the other two severity categories we simply put them in the queue on that day with the indicated priority. It’s up to the programming staff to take things off the queue by priority, first come first served within any priority group. Once a programmer is assigned to fix the problem, it shifts to the “**Assigned”** stage, which means it is in the process of being fixed”.
3. It isn’t unusual for different customers to report problems that appear to be similar. If I find that a problem seems to match a previously reported problem, I link the new problem to the previously reported problem, and it takes on the same status and priority as the earlier problem. What’s supposed to happen is that when they fix the earlier problem they should also test that the new problem is also resolved. (Sometimes the programmer overlooks the linked problem and I have to point that out to them.) In theory, a linked problem should be fixed sooner because the other problem was reported earlier. What happens sometimes (about 1 time out of 4) is that the programmer discovers that the new problem and the old problem are not the same. (That doesn’t happen until after the programmer actually starts to work on the old problem.) In that case I unlink the new one and it becomes a stand-alone problem, going into the queue that day.
4. Once the repair has been made, the problem report moves to the “**fixed”** stage, which means it is waiting for regression test. After regression test is complete, it moves to the “**tested”** stage, which means it is and waiting for final approval. The CCB (see next step) is responsible for final approval. Perhaps one time in four, the software fails the regression test, in which case it goes back to the **“waiting”** queue, but is given top priority and flagged so that it will be assigned to the same programmer as soon as they have finished their current assignment.
5. The other thing I do every morning is look for which problem repairs have been approved the previous day. Repairs must be approved by the configuration control board (CCB), which consists of Rachel, Sharleen (the maintenance programming manger), and the quality assurance manager (I forget that name – they just hired someone new for that position). Once a repair is approved, I wait for Sharleen to prepare the update installation package. This usually takes only a day or two, after which she will change the status of the problem report to “**waiting for shipment**” on the tracking system. When this happens I send an email to the customer representative, who downloads the installation package. Once the rep reports a successful installation, it is moved to an archive, where we keep track of all previous fixes in case we need to see what happened, who fixed it, etc.

How Long it Takes: For a high priority problem, it typically takes a minimum of 2 months and more typically 3 months between the time I put it in the queue and the time the approved repair is complete. I don’t know why it takes so long. Normal problems may take a month or two longer than that and minor ones may take even longer than that.

Their Assessment:

* I don’t know why there are so many problems with some of the applications. The programmers sometimes claim it’s because they are larger, have more code, and thus we should expect more defects. But the defect rates on some products are way too high and I think there’s more to the problem than their sheer size.
* Regarding the delays in fixing defects, I think there are several problems:
  + One is that when I unlink a problem it goes into the queue for that day rather than for the day when I originally linked it. That can easily add several weeks to the overall timeline for that problem report. This is the way I’ve always been told to do it and there’s no way I can give it the original date because our problem report tracking system assigns the date.
  + Another problem is that the CCB and regression testing steps sometimes take a lot longer than I think they should take.
  + Also, I can’t understand why the programs fail regression test as often as they do. We have very good programmers in the maintenance department.

Interview Report

Person Interviewed: Rachel Wallace

Job Title: Maintenance Manager

Description of job: Rachel is George Wilson’s supervisor and the maintenance programming and maintenance test teams also report to her. Her job is to resolve priority disputes with problem reports, to assure that there are adequate people and resources for testing of all repairs, and to assure that everything is ready before shipping updates back to the customer reps. In conjunction with the programming manager, she assures that there are adequate programming resources for handling the level of problems being reported. She also sits on the configuration control board (CCB), whose job is to approve all proposed repairs before they are started (to make sure they don’t cause problems or interfere with other repairs) and to give final approval after repairs have been made (to assure that all repairs were made correctly).

Failure Problems: I am very frustrated by the high failure rates on some of our products. The software development department should be interviewed about this. Sharleen can give you some data to show which programs are having the highest failure rates.

Failure Report Processing: I am involved in several ways, in addition to overall supervision of the people who do the clerical work and the testing. I do the following:

* *Resolve any disputes* about what priority should be assigned to a problem repair – I do this by conferring with the maintenance programming manager and the customer rep. We usually come to an agreement rather quickly, but it sometimes takes us a few days to find a time when the programming manager and I can be in the same place at the same time and can get the customer rep on the phone.
* *Chair the Maintenance Configuration Control Board (CCB)*. There are three points where the Maintenance CCB is involved:
  1. **Repair plan approval.** When a repair has been assigned to a programmer, their first assignment is to assess the problem and present a plan for how to fix it. The CCB must make sure their proposed repair doesn’t cause problems, such as conflicting with some other repair that’s going on. Such conflicts don’t happen very often, but when such a conflict does happen the CCB’s review sometimes prevents a major catastrophe. If we discover this kind of conflict we ask the programmer to figure out a different solution that doesn’t involve a conflict and present it to us at our next meeting. Sometimes, the solution is to simply wait until the other repair is complete.
  2. **Preliminary repair approval.** When a repair has been completed, the CCB reviews the programmer’s solution and unit test results to make sure the repair was made according to the plan. This step is necessary because the programmer occasionally tries to go beyond the approved repair or overlooks some important part of the repair.
  3. **Final repair approval**. After the regression test is complete, the CCB gives final approval. This is usually very straightforward but occasionally the regression test uncovers a further problem. We decide what to do in that case. The purpose of a regression test is to make sure that a problem repair doesn’t cause problems in other parts of the software. A regression test (which can take several days to run) consists of a comprehensive set of tests on the software – it tests everything, not just the problems being fixed. We typically run regression tests once every few weeks and include all problem repairs that have been completed during that time (because regression test is expensive and takes a long time). If the regression test uncovers a problem, it usually takes a while to figure out which problem repair is the cause. That repair is sent back to the queue with top priority so it will be the next item assigned to the programmer who originally made the repair. Any other repairs that were part of the regression test suite have to wait until that bad repair is fixed so we can re-run the regression test.
* *Authorize sending the update back to the customer*. This happens after the final CCB approval. The main thing I do here is make sure the actual update installation package has been assembled correctly by the test team so the installation of the repair will go smoothly.

How Long it Takes: Other than what was mentioned above, the main point I would make is that the CCB, which consists of myself, the maintenance programming manager and the quality assurance manager, cannot always meet when we would like to. We normally have a weekly meeting every Wednesday afternoon. If one of us cannot make it they send a deputy. That works for all but the final approval because our process requires that all three of us sign the final approval paperwork on the same day. This restriction was instituted several years ago to assure that all three of us were actually there at the same time to discuss any issues. Back then we didn’t have smart phones or email, so meeting face-to-face was considered important. I think we could come up with a more efficient approach today.

Their Assessment: Our process involves a lot of steps and a lot of possibilities, so it is somewhat complicated. Any problem report can sit in the queue waiting for other things to happen. For example, regression test is sometimes only performed when multiple problems have been resolved so waiting for regression test can cause a significant delay. I suggest you talk to Wendy, the Maintenance Test Team Leader, who reports to me, for further details and suggestions.

Interview Report

Person Interviewed: Sharleen Jefferson

Job Title: Maintenance Programming Manager

Description of job: Supervises the programmers and assigns them to work on specific problem report repairs. Also sits on the CCB.

Failure Problems: Certain programs cause us a lot of trouble because they have unusually high failure rates. The ones we are having trouble with right now were all produced in the last three years. Fortunately, we’ve collected data during that time. See the ***IPC Defect Data*** file for the measurements we have made and other data about all the software programs we have produced in the past three years.

Failure Report Processing: I’m mainly involved in these ways:

* *Handling exceptional situations*
  + The exceptions include things like sending someone to the customer’s site in the event of a severe problem and disputes about what priority to assign to a problem report. These typically don’t take long to resolve, but sometimes they result in shuffling priorities around. If a new problem ends up with higher priority than some older ones, the older ones tend to get delayed longer. Unfortunately, the people who set priorities for new problems don’t ever tell us which of the older problems to set aside to make room for the new ones so I have to make that judgment on my own. It puts me under a lot of pressure sometimes because I’m being criticized by customer reps and sometimes by people higher up in the management chain, but there’s nobody higher up I can turn to for resolving the priority. When three people all demand top priority, there’s nobody else to decide which of the three wins.
* *Assigning programmers to problem reports* 
  + We have over one hundred software products being maintained, with about ten new ones each year. Fortunately, most of the ones with problems are fairly recent – we’ve fixed or replaced most of the older ones. I only have a few programmers, and my programmers are usually not all that familiar with the software they have to repair. However, some of them do specialize a bit in certain areas. For example, one of my programmers is an expert on data bases and another on HTML. Normally I try to assign the most appropriate programmer to each problem report.
  + It’s fairly common to have more than one failure report for a given software product, especially the products with high failure rates. In that case I usually assign them all to the same programmer. I do this because it takes several weeks for the programmer to understand the software well enough to debug it. In this way the programmer is already familiar with the software when working on the later problem reports.
  + Sometimes the best programmer for the job is busy doing something else and we have a policy that says once assigned to a particular problem the programmer cannot be pulled off to work on another problem. I don’t know where this policy came from but it is sometimes a real thorn in my side when the programmer is working on something unimportant and there’s a priority problem that they are best suited to. And most of my programmers could multitask well enough to handle more than one assignment, if they were allowed to do so. [Note from DJF: I learned since that this policy was instituted by a former company vice president who strongly believed that no programmer should ever be asked to multitask. That vice president has since moved to another company.]
  + Another challenge I have is that sometimes a programmer has a scheduled vacation or unscheduled illness or family emergency when he or she is in the midst of working on a problem. I try to factor the scheduled vacations into their work assignments, but sometimes the repair takes longer than I planned and a half-done repair must wait while the programmer goes on vacation or takes a holiday. The unexpected absences are difficult to recover from because if I have to assign a new programmer they have to start all over. Learning how the program works takes a long time because of the poor documentation we get from the original product developers. Their “agile” development approaches doesn’t produce much documentation to help someone else who has to do maintenance on the software a year or more later. Our maintenance approach is also focused on rapid software development, so we don’t produce much documentation during maintenance. This is why it can take a long time if a new person must take over a maintenance or repair assignment. This affects about four out of ten problem repairs, sometimes adding 2-3 weeks of delay. (In the old days we produced more documentation and this wasn’t so much of a problem, but when the company switched to “agile” methods, a lot of the documentation went by the wayside. I’m not sure that this has helped much in terms of improving the overall time it takes to fix problems.)
  + The original development team uses pair programming so that there are always at least two people who understand each software module. However, in the maintenance organization we don’t have that many programmers and our work is usually relatively straightforward, so we don’t do peer reviews or inspections or pair programming.
* *Serving as a member of the CCB*.
  + Your notes from Rachel pretty well cover this except for one issue:
    - In order to discuss a repair sensibly, at any stage of the approval process, we need to speak with the maintenance programmer who is making the repair and, on many occasions, with the development programmer who originally wrote that part of the software. Those original programmers are not always available during the scheduled CCB meetings because they work in an entirely different part of the company (Software Development). We need to see them fairly often because IPC’s “agile” development style tends to produce very little documentation and sometimes the code is difficult to understand (although it is supposed to be well commented). The irony is that sometimes the original programmer doesn’t understand his or her own code, especially if it was written more than a year before. So, as a result, we sometimes have to ask the maintenance programmer to try more than one approach or to make assumptions that later turn out to be wrong and require rework on the repair.

How Long it Takes: In my comments above, I’ve pretty well outlined how long it takes for the various things I have to do.

Their Assessment: You’ll have to ask Melinda over in Software Development about why certain products have higher failure rates. Regarding the maintenance process, I mentioned several things in the discussion above. I haven’t given a lot of thought to which of them has the greatest impact, but I’d suggest that there are a lot of things that contribute to the delays and what we really need is to rethink the whole process. I’m looking forward to seeing what your students come up with because I haven’t taken the time to analyze the whole picture.

Interview Report

Person Interviewed: Narayan Bhat

Job Title: Maintenance Programmer

Description of job: Works for Sharleen, the maintenance programming manager. His job is to analyze problem reports, identify the source of the problem, and make the required repairs. He often presents to the CCB but is not a member of it.

Failure Problems: I can vouch for the fact that some products have high failure rates – I’m the person who has to fix them! I’ve become an expert on Zillow and Justify, both of which deal with data base issues (my area of expertise).

Failure Report Processing: I am typically assigned to work on a problem by Sharleen. Usually I am assigned to one problem at a time, which I think is a mistake because sometimes I need to wait for some information or for a test to be performed or something else and I think I could work on a different problem in the extra time I have available.

Nevertheless, here are the steps I go through:

* *Look at the problem report and try to reproduce the problem*.
* *Debug the software until I figure out what’s wrong*
  + Because of the skimpy documentation in our way of developing software, this often requires that I consult with the programmer who originally wrote the code. They write a lot of comments in the code, but that only explains how the code works. Sometimes I need to understand more about the design or requirements. The original programmers aren’t always readily available because of priorities in their own organization. (They are usually under intense pressure to meet product completion deadlines, so they don’t get back with me as soon as I would like.) This is an example of the kind of delay I mentioned above.
* *Propose a solution and present it to the CCB for approval*.
  + Another source of delay because they normally only meet once a week. They will meet sooner if it is a high priority problem for an important customer, but that isn’t needed very often.
* *Implement the solution and unit test it*
  + The unit tests developed by the original programmer are usually not available because we don’t save them. So I have to come up with new unit tests. This takes time. (I have a collection of test programs I’ve developed for the applications I fix most often). I also don’t have the original requirements or design documentation, so sometimes I overlook important things that should have been tested.
* *Present the solution and unit test results to the CCB for approval*
* *Give the software to the test group to perform system tests and regression tests*
  + At this point I start work on another problem, but occasionally they bounce it back to me. When that happens, I’m supposed to finish my new assignment before going back to make whatever change is needed on the old problem.
  + Once in a while the test team finds a problem that I overlooked. When this happens, since I’m busy on another problem, they put it back in the queue with top priority so that when I’m done it’s the next thing I’m assigned to do.
* *Deal with linked problems*.
  + Occasionally the problem I’ve fixed is linked to other problems because the maintenance clerk thought they had the same solution. He’s often right about that, but not always – I sometimes wish he had consulted me before linking the two problems, although in the absence of good documentation I’m not sure I could have made a better decision than he did. Anyway, if I determine that they are linked I have to test to make sure both problems are fixed. If I determine they are not the same problem I tell the maintenance clerk, who unlinks them and reschedules the newer problem.
  + Once in a while I don’t notice that two problems are linked. The problem is with our tracking system. It isn’t obvious that they are linked. There’s only a small, hard-to-read flag, which indicates whether or not my problem is linked to another one. (The maintenance clerk usually lets me know if I’ve missed something like this.)

How Long it Takes: Once I start on a problem it usually takes me about two weeks if I don’t need to consult with the original programmer (that happens about half the time), but I think I could do it less than a week if I had better documentation. If I have two problems with the same software application (or one of the high failure rate applications that I’ve seen before), I can usually do things faster because I’ve already figured out how the program works. If I have to consult with the original programmer, add another week minimum and sometimes two. Then I have to develop unit tests for the code, which often adds another week if I haven’t tested the application before. After unit test is complete, I send it on to the test group but if the test group discovers a problem in system or regression test, they send it back to me. Because I’m working on something else by then, I don’t get back to the problem report until I’m finished with my current assignment. This can take a few weeks.

Their Assessment: I blame the high failure rate problems on the software development group – you’ll have to talk with them. As to the maintenance delays, I’ve already mentioned several ways to speed things up. If I had to name one thing, it would be the fact that our process and our policies often leave me with idle time – this seems very wasteful to me.

Interview Report

Person Interviewed: Wendy Stottlemeyer

Job Title: Maintenance Test Team Leader

Description of job: Wendy reports to Rachel (maintenance manager) and her job is to perform system level tests on all repairs to see if they work, to run regression tests, and to prepare final update packages for completed and approved repairs.

Failure Problems: I don’t know why, but I see certain programs a lot more often than average, so I assume they came out of software development with a lot more defects. For example, just this week we’re testing Halo and Zorro for about the third time this year.

Failure Report Processing: My team’s main job is to *run system level tests* on the software after it has been repaired and unit tested by the programmer. We do this by taking the problem report and the unit tests developed by the maintenance programmer, producing our own independent, system level tests, and running those tests. Our biggest problem here is the lack of a good set of requirements or system tests for the software modules that have been repaired. The maintenance programmers always have a meeting with us to explain how the program works and to suggest some system test approaches. They tend to produce unit tests based on the way the code works, not based on its original requirements, in part because in most cases neither we nor they have any documented requirements. The requirements and the tests used by the original programmers are rarely available to us in the maintenance group. Because we in the test group are independent of the maintenance programmers we sometimes find mistakes they overlooked (maybe one time in three). I think it would help if we had joint peer reviews or inspections of the code along the way. This would also familiarize us with the software a little sooner. (As far as I know the maintenance programmers don’t do peer review or inspections because those are perceived as taking too much time and, also, because usually only one of the programmers understands the code being repaired.)

Our other job is *regression testing*, which is running a large, comprehensive suite of tests on the complete, final software package, with the repaired modules included in place of the ones that had errors. The regression test software is the only software we are given from the original program development team, and we sometimes have to adjust it because new features have been added since then, but at least we have the regression test software as a starting point. We schedule regression tests only when all current repairs on that particular software have been completed. We do this because, on a typical product, the regression test takes a whole day or sometimes two and requires our whole staff – it’s a big deal. If someone fixes a module and there are three other modules being fixed on the same software product, we typically wait for all of them to be done before running a regression test. This can cause a several-week delay in final approval of a repaired module.

How Long it Takes: It takes us 2-3 weeks to develop and run a typical set of system level tests. We could do this in 1 week if we had good, system level test code and requirements from the program’s original developers. (When we update these system tests and regression tests for new features, we keep them archived to help us the next time we test these same applications.) Regression test is intended to be comprehensive, and it’s one of the reasons why our final updates are usually very good, so I think it would be mistake to water it down to make it less expensive or comprehensive. However the policy for how often to run regression tests is worth looking at. There was a time when regression tests took a whole week, and the policy about how often to run them was developed back then. Computers have gotten a lot faster since then and we can do them in a day most of the time.

I’ve mentioned some other sources of delay in my comments above.

Their Assessment: Managers are often complaining about the time it takes us to complete our tests, especially our regression tests, because we are the last ones involved before shipment back to the customer representative. But from what I can tell, there are plenty of delays before we even see the software. I wish we could insist that they make the code more testable – we often have to work hard to insert stubs and drivers and other things to make it possible to actually test it. The lack of good documentation and test code also slows us down.

Interview Report

Person Interviewed: James Johnson

Job Title: Quality Assurance Manager

Description of job: Supervises QA staff and serves on both software development CCB and software maintenance CCB.

Note: James was hired about two months ago to replace a previous QA manager who took a job at another company.

Failure Problems: I’ve only been here for a few months, but I’ve noticed that some of the products we produce seem to have abnormally high failure rates. I’m still trying to understand more about the software development process so I can focus in on the cause of this problem. I’m hoping your DMAIC study will help me.

My staff members support both maintenance and development organizations and they tell me that the high failure rate problem did not appear until the company switched to agile methods back in 2015. However not all software has these problems, so I’ve heard a lot of hypotheses about the cause and seen very little evidence to support any of them. Some claim it’s the programming language, some the particular style of agile development (we use SCRUM and Extreme Programming), and some blame the programmers, although we hire only very good programmers.

Failure Report Processing: My perspective is from two activities: my role on the development and maintenance CCBs and my role in quality assurance for both product development and maintenance.

* *Configuration Control Boards (CCBs)*

I am a member of the software development CCB and also the software maintenance CCB. My job on both CCBs is twofold:

1. to make sure they follow IPC’s processes and policies, and
2. to contribute to the discussion and decisions when the issues are complex. I’m sometimes the tiebreaker, but we usually seek to achieve consensus of all CCB members.

The maintenance CCB only has three people on it, whereas the software development CCB has four members.

My primary concern is to assure that we ship quality products. Both CCBs have well defined, if somewhat bureaucratic, processes that have been formed over the years. As I see it, the process we follow for software development is effective and the agile approach has made it somewhat more efficient than the approach used in maintenance. The process for responding to customer problem reports is effective, but inefficient. By the time we actually install a completed or repaired software component in the customer’s site, it almost always works well because our process has eliminated most defects along the way.

The Development CCB is only used when the product is deemed to be complete by the development staff. Before the final approval step, the programming staff is responsible for making sure things are done correctly. They do not do formal tracking of problem reports during product development, only during final system test. The Maintenance CCB is involved three times in the processing of a problem report, so we tend to have a somewhat better perspective on the overall situation. We also have on-line access to the status of all problem reports, so we can track any of them if we wish.

Regarding delays in fixing customer problems, I’d say that, if we exclude the critical problems, our average problem takes about 3-4 months to get through the system. This seems long, but there are a lot of steps and, more importantly, several points along the way where problems are not being worked on (waiting for some reason or other). I don’t have enough experience at IPC to know exactly what’s holding things up most of the time. Our process is rather complicated - we need to take the time to analyze the whole thing - so I’m glad you are doing that with your DMAIC analysis.

Getting back to the software development CCB, I see some things that could be done to improve coordination between the development and maintenance organizations. For example, when the development staff has completed a project they retain only the source code and the regression test suite. There’s little other documentation except comments within the code itself. Whatever code was used to unit test or system test the original software is discarded. The maintenance staff could be more efficient if they had such information, but the development staff considers it to be too much of a burden. They also contend that requirements and design are so fluid in their agile development environment that it’s a waste of time to document them before the software is complete, and after that there’s no justification to do so – they have other projects to work on.

* *Quality Assurance*

In my QA role I mainly observe the programming and testing operations to make sure people are following our processes and procedures and doing a good job. QA does not perform testing – we watch what the testing team does and make sure they do it correctly. We also watch the programmers and make sure they do their job correctly. A lot of our time is spent coaching them – explaining what they are supposed to do. This is especially true with new hires, who don’t understand the process as well as the more experienced programmers and testers. I think all the people are very capable, and a lot of the mistakes that occur during maintenance are the result of our limited documentation. However, I also think it would be helpful if there were some sort of peer review or inspection process for the maintenance activity. There aren’t very many maintenance programmers but I think some of the test staff could review the work before it is finished rather than waiting until it has been officially turned over to the test group.

How Long it Takes: As I said above, it takes an average of about 3 months, with lower priority problems sometimes taking significantly longer. I’ve seen it as short as 1 month when management pushes it through, and as long as 8 months when it’s a low priority problem.

Their Assessment: Many of the rules we follow in our CCBs were put in place to deal with problems in the past. As a new QA manager, I’d like to re-write some the rules today using alternative approaches because we have better tools and much better communication mechanisms. See my other suggestions above.

Interview Report

Person Interviewed: Melinda Shah

Job Title: Software Development Manager

Description of job: Supervises all software development for new products. Is a member of the software development CCB.

Background: About three years ago we switched our development approach to use “agile” techniques. I hired Alan Andrews to help us make this transition and to take over for a software manager who retired. He’s a very smart guy and the transition has been remarkably effective. Since that transition, about half of our projects use SCRUM and half use Extreme Programming (XP). I don’t know if there is a significant difference between them, but the switch to these approaches has improved our new product development productivity by about 25%, which is good from James’ perspective because it improves the efficiency of the development team. Part of that improvement is due to the fact that we don’t generate as much documentation as we used to. In particular, the agile methods call for us to use a rather informal approach to requirements (lots of meetings with customers but limited and informal documentation of the resulting requirements). The programmers know what to do and don’t want to “waste time” by writing it down. They argue that a major advantage of agile techniques is ability to deal with uncertain, unclear, and changing requirements, so they don’t want to document a lot of things that will probably change by the time the project is complete. We also don’t spend much time documenting designs – with the agile methods we refactor our designs fairly often and we don’t want to have to keep updating the design documentation. (We do comment our code well, I believe.) I mention this because the maintenance people claim that lack of such documentation is slowing down their response to customer problems. We’re both so busy (my programmers and theirs) that we’ve never gotten together to see if we could find a compromise solution. My job performance is judged by how well I meet deadlines for new software products, not by how quickly we respond to customer problem reports.

Failure Problems: I’m aware that some of our more recent products have had abnormally high failure rates and we’ve been collecting data for three years to help us better understand our situation. I understand that Sharleen Jefferson has provided you with a file containing three years of this data. I’m hoping you can help us identify the causes of our failure problems because my managers and programmers have different opinions as to what is causing the problems.

Failure Report Processing: I’m rarely involved with the processing of customer problem reports, although people on my staff are often asked to help the maintenance programmers. When I get a request for such a consultation I have to balance the programmer’s current job priority with the severity of the maintenance problem. But we usually respond within 1 week to such requests. Personally, I think that we should give higher priority to support problems than to meeting deadlines for new software development, but my boss, Jeff Donahu, insists that we meet new product development deadlines. He seems to focus on making more profit from new software products. As it is, the cost of developing new software products is often barely covered by the revenue we generate from their sale, so Jeff is always pushing for my group to be more efficient.

Another thing the maintenance people complain about is lack of unit test and system test code. This is again due to our agile techniques. We put a lot of effort into producing a comprehensive suite of tests that they use for regression tests, but our unit testing and system testing are very informal, often being off-the-cuff approaches by our software developers, and we don’t have the time or the budget to formalize those tests.

How Long it Takes: As I mentioned above, it takes us a week or less to respond to a request for help from the maintenance programmers.

Their Assessment: I don’t see how we could help them improve their response time without making our job more time consuming, which would not please my boss (James). However, we should try to figure out what is causing the abnormally high defect rates. We’re counting on you to give us some insight there.

Interview Report

Person Interviewed: Alan Andrews

Job Title: Manager of Financial Products Software Development

Description of job: Manages the product development team for financial products. Is a member of the software development CCB.

Background: Although my group is called Financial Products, we actually develop software for a wide range of application areas. Some of our products are for web sites, some for data bases, and some for heavy-duty data processing. We also do a lot of joint projects with the Payroll Products group (managed by Walter Wilson). I came to IPC three years ago as part of the switch to agile methods. Walter’s been at IPC for over twenty years and is a bit more “old fashioned” about software development, although I have to say that he has done an excellent job of adopting and implementing the agile techniques. Walter and I share the same programmer pool, but we tend to prefer different people for our projects – my philosophy is to use the smartest programmers I can find, whereas Walter often seems to pick people that I see as competent but not quite as fast. As a result, my projects usually come out on schedule more often than Walter’s do.

Failure Problems: Some of our products have high failure rates. We have a lot of different opinions about why this is the case but have not really identified a specific reason. Maybe your DMAIC analysis will help us identify the cause or causes.

Failure Report Processing: The programmers on my staff are often helping the maintenance staff. We could probably do this in a more streamlined manner, but my group is under intense pressure to meet product development deadlines so working with maintenance is not my top priority.

Their Assessment: We need help and we’re under too much pressure to meet deadlines to have the time to do our own analysis.

Interview Report

Person Interviewed: Walter Wilson

Job Title: Manager of Payroll Products Software Development

Description of job: Manages the product development team for payroll products. Is a member of the software development CCB.

Background: We develop software for corporate customers, mostly in areas such as employee benefits, payroll, taxes, accounts receivable and accounts payable. We do some of these jointly with the Financial Products group. I’ve worked for IPC for almost 25 years and have seen software development methods and languages come and go. When I started we were still using Cobol although now we’re mainly using Java and C++. About three years ago we switched to agile development methods because our CEO wanted us to develop software more quickly. We brought in Alan Andrews to replace a man who retired and to help us make the transition. Initially, I didn’t favor this approach because it’s not how I learned to do things. However I’ve developed a lot of respect for how agile techniques can help deal with unstable requirements. That’s often been a problem with past projects. In the first few months of 2015 I spent most of my time learning about agile methods, so Alan managed the new products we developed at the beginning of that year. But I started managing my share of new projects in March, so you’ll see my name on more or less an equal number of projects with Alan in products that we released in the second half of 2015 and in later years.

I remain quite concerned about product quality and, as you’ve been told, our product failure rate has started to deteriorate. But, strangely enough, it seems to happen only with some of our products. So I suspect there’s a factor we haven’t yet identified that’s causing this. Alan and I each supervise software development on different projects, but we draw our developers from the same programmer pool (and we jointly select new hires). He’s a relatively new manager here and he taught all of us how to do agile development well – I admire his skill and talent. But we have different ideas about what characteristics to look for in our developers. He tends to prefer “hotshot” programmers who are very skilled, but don’t always know much about the applications they are writing software for. My most important criterion is application domain expertise. I pick programmers who have experience writing the kind of software we need for the kind of customer we are serving. We may not get the job done quite as fast as Alan’s projects do but I’m usually very confident that our software will satisfy the customer.

Failure Problems: You’ve heard about our concerns regarding high failure rates. Some of my programmers think this is due to our switch to agile development, and some believe its due to the languages we use. I wonder whether SCRUM (which we use on about half of our projects) produces notably better or worse results than Extreme Programming (which we use on the other half). I hope you can examine that more closely. We never kept good metrics until three years ago so we don’t have good data on the non-agile projects. But it is clear that our recent projects tend to vary quite a bit more than they used to in terms of defects. I think this could be due to the increase in average program size. Some of our applications are over 100,000 lines of code. Maybe your DMAIC analysis will help us identify the cause or causes of these failure problems.

Failure Report Processing: Our programmers try to help out the maintenance staff as much as we can because we know this is crucial to keeping our best customers. But our CEO insists that we keep focused on meeting our development deadlines.

Their Assessment: We’ve changed. We are writing bigger applications that take larger teams of people and our defect levels are not consistent from product to product. I hope you can help us figure out how to improve that because our future depends on it.

Interview Report

Person Interviewed: James Donohu

Job Title: President and CEO

Description of job: Chief executive officer of IPC.

Background: We’ve been in business for over 25 years and have a strong reputation in our areas of expertise. Today we produce about ten new products each year. We’ve managed to keep up with the rapid changes in how software is developed, delivered, and supported. I try to keep all operations running efficiently at IPC, and I’m proud that the software development group has reduced their costs and schedules significantly by switching to agile techniques. That has resulted in significantly increased net revenues from new product sales. But we seem to be losing money in terms of increased maintenance and support costs. Even worse, some of our biggest customers have experienced significantly more failures in a few of our recent products and they are letting us know about it.

Failure Problems: I’m really getting frustrated with the dismal failure rates on some of our products. And what’s especially frustrating is that this doesn’t happen with all of our products. We have to fix this problem. We’re in danger of losing customers because of this.

Failure Report Processing: We make good money from the maintenance fees paid by our customers, but we’re at risk of losing customers because of our slow response to customer problems. I hope you can figure out what the problem is with the maintenance group.

Their Assessment: I wish I knew what the causes of these problems are, but because I don’t I’m asking for help from you.