Proceedings of LISA '99: 13th Systems Administration Conference

Seattle, Washington, USA, November 7–12, 1999

ORGANIZING THE CHAOS: MANAGING REQUEST TICKETS IN A LARGE ENVIRONMENT

Steve Willoughby

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Organizing the Chaos: Managing Request Tickets in a Large Environment

Steve Willoughby – Intel Corporation

ABSTRACT

The Intel Performance Microprocessor Division Engineering Computing (PMD EC) team consists of 85 systems administrators serving the needs of 2,000 engineers while trying to manage 7+ terabytes of data and 2,500-3,000 computers. Without some kind of task management strategy, the needs of so large a group would quickly spread chaos and confusion. In fact, due to the nature of our relationship with our "customers" (PMD chip designers and other related employees and contractors), who have strict and demanding requirements of their computing environment's features and stability, extra discipline is required just to keep on top of our issues, and to properly prioritize and execute them.

The EC group receives several hundred customer help requests each week. To organize and track these requests, ensuring that they are done in the right order, none are forgotten, and that our customers are kept informed as to our progress, we have evolved a set of tools, collectively named "REQADM" (for "REQuest ADMinistration"). This tool suite allows customers to submit requests, which get assigned to EC technicians to be worked on, and eventually brought to a resolved state.

This paper describes the REQADM tools, the organization we evolved to support our customers, and the lessons learned along the way.

Pre-History

We started with a system similar to Trouble-MH [3], QMH [7], Req [2], QMH-with-complainers [9], and Request [8]. We set up E-mail aliases with names like "sys-help" and "hot-help" (the latter for requests deemed – by the customer – to be more urgent than normal). These aliases automatically dumped messages to MH-style archives, thereby effectively assigning a number to the "request ticket".

On the way into the archive, the ticket got stamped with some administrative information at the bottom of the message, which looked something like this:

```
XXX Admin Information Follows XXX
NUMBER:    23669
DIRECTORY: sys-help
RECEIVED:  Fri Feb 9 11:00:58 1996
```

The file itself remained in RFC 822 [1] format, with mail headers intact and the request text and administrative information as the mail message body.

An EC administrator (called the "request dispatcher") would have the task of monitoring the incoming "queue" of requests. This would be done by running, as often as possible, a Perl script to search all the files in the *sys-help* and *hot-help* directories, scanning their contents to see if any had been assigned to an owner yet, and printing out a list of any which had not.

From this list, the dispatcher would run a Perl script called *assignrequest*, assigning each new ticket to an EC administrator who would "own" the ticket and be responsible for seeing it to completion. This would cause a set of text lines to be appended to the ticket file:

```
ASSIGNED TO:      steve
ASSIGNMENT DATE: Fri Feb 9 ...
ASSIGNED BY:      vicki
```

Now, the user "steve" would be responsible for the ticket. An E-mail notification would be sent by *assignrequest* to the original submitter and the new owner.

There was a large set of shell and Perl scripts used to manage the ticket in their MH-format files: *noterequest* to add a note (status update, commentary or question), *reassignrequest* to change the owner, *request.nag* to send nasty notes to EC admins who had requests getting old, *showrequest* to view the contents of a request, *whoserequests* to scan the queues to see who owned what requests, and so forth.

Problems With the Old System

We ran into a few problems over time, which made us think we should reconsider how we organize our team and our ticket tracking software.

First of all, since the files were nothing but MH mail archives, they were vulnerable to a host of mishaps. Bugs crept into the scripts as we added hack upon hack, some of which corrupted our ticket files. For example, if a script had to reassign a ticket to a new owner, it had to add "ASSIGNED TO:" lines to the bottom of the file (no lines were ever deleted, so we always had a record of what happened to a ticket):

```
ASSIGNED TO:       fred
ASSIGNMENT DATE:   Fri Feb 9 ...
ASSIGNED BY:       steve
NOTE:
I'm assigning this to Fred, since
I'll be out of the office for a
few days. He'll be taking over
for me.
```

Since our scripts relied on being able to use *grep*(1) to search for keywords like "ASSIGNED TO" to see who the current owner was, the new script had to know to munge the previous assignment(s), so they wouldn't be mistaken for the new one:

```
ORIG-ASSIGNED TO:      steve
ORIG-ASSIGNMENT DATE: Fri Feb 9 ...
ORIG-ASSIGNED BY:      vicki
```

As the system features expanded, and the number of management scripts multiplied, it was inevitable that a new script would corrupt data that an old script assumed would be in a certain format.

There was also the risk that user data or admin comments (see the "NOTE" added to the re-assignment above) would contain text which would look like an administrative keyword like "ASSIGNED" or "CLOSED". As new keywords were added by new scripts, old tickets which didn't break before suddenly caused problems.

There was little possibility for control over who was allowed to edit tickets, since everyone in the EC team had to have write access to the files to run MH commands and ticket-management scripts on them.

And of course, every so often, someone would forget that he or she was in their ticket "folder" and issue an MH command to re-pack the folder, effectively renumbering all the tickets in the system (duplicating numbers of older closed tickets). At other times, the MH sequence files would get corrupted and all incoming requests would end up with ticket #0 (effectively going nowhere).

The scripts were getting fragile as their interdependencies (and even their own internal code) became more convoluted and difficult to maintain. The old system was collapsing onto itself, and needed serious redesign.

Customer Relations Issues

Some of the problems we faced with this previous model were not completely related to the software tools being used. No computing support group operates in a vacuum, and the way we interact with our customers is an important factor to be considered in evaluating and redesigning a support system.

For one thing, it can be advantageous to have one's ticket queue visible to the user community, as I discovered at the company I worked for prior to Intel, where I was the sole systems administrator. I implemented a simple ticket tracking system there to help my users understand how many issues I was dealing with at any given time. This made them far more sympathetic to the fact that each of their requests were to be prioritized and some would have to be denied or deferred for lack of available time to spend on them.

On the other hand, this same feature is a bit of a double-edged sword. We are also aware that whatever problem is preventing a customer from getting their work done is extremely urgent to them, and it may not seem relevant at that moment that there are other problems elsewhere in the department that also require EC's attention. This can be frustrating to all concerned. We wanted to make sure that the ticket tracking system we migrated to would help our customers see where their issues fit in to the bigger picture. This would help them understand what priorities we need to juggle for the common benefit of all. It would also help EC be constantly aware of what the "hot" issues are, to be sure we are always working on the right issues in the right order.

One of the problems with our MH-based system, which made this situation difficult to resolve, was the fact that the system only kept a single bit of resolution for ticket priority: it was either "normal" or "hot". Since this did not provide us with enough insight to adequately judge relative priority between tickets, there were times when EC technicians would mis-prioritize jobs, frustrating our customers, and times when customers would file too many requests as "hot", diluting the meaning of that priority.

This is actually a combination of technical and political (managerial) issues, and we approached solutions along both of these lines, as we'll discuss below. From a technical standpoint, however, one thing that was clear to us was that any new software solution needed to include features to give the customer more control to specify true and accurate priority.

Another problem we faced was that there was some concern that tickets were occasionally closed before the problems they described were fully resolved. This was not a major problem for us, but it is the kind of thing that doesn't need to happen very often to become a customer relations issue (particularly if it happens to a key customer).

While there may have been isolated instances of less-than-careful work on the part of a sysadmin, or a customer feeling uncomfortable with the actual solution offered, the chief source of concern here was communication between EC and the customer. Most often, when this happened, the real issue was simply that the sysadmin didn't understand exactly what the customer needed, or the customer didn't fully understand the nature of the problem, and therefore that the solution offered would really solve it.

This pointed out to us that we needed to implement a reliable mechanism within our new system which would formalize – at least – a communication loop for final approval of a problem's resolution to be certain that all was resolved to everyone's satisfaction.

Some Attempted Improvements

While the new request system was being designed and written, the computing group made a number of other attempts to improve our situation. These included public relations, process, organization and software improvements.

Customer Surveys

It is important to the success of a support organization to know how well they are perceived by their customers to be meeting their needs. The best way we are aware of to get this information is to ask them for feedback.

This is not as straightforward as it sounds, however. Our customers, being designers and engineers, are usually very busy at work inventing and improving products for our company. We want to take up as little of their time as possible to perform "overhead" paperwork like customer surveys. (We would also like to take up as little of our time as possible to administer and analyze these surveys as well.)

The old software had no automated features for polling customers to see how they felt about our service, so our attempts to assess their satisfaction level were manual and occurred at irregular intervals.

Initially, we tried the "broadcast spam" method, sending a general survey by E-mail to everyone we supported, asking them to reply with the answers to our questions. Only about 5% of our customers responded to the surveys, and most of them were our more vocal users, from whom we were already hearing feedback anyway.

Later, we tried the "barbarian invasion" method. One of our key front-line support people would randomly select a few tickets submitted each week, and go sit down in the submitter's cubicle for five minutes and interview them to get their feelings about how well their issue was handled. This was more costly (in terms of time spent) to implement, but ultimately was more successful in gathering feedback from a wider sample of our user population.

We eventually decided that our new tracking system would need to incorporate some capability to poll customers who are using it, asking them a few questions about how well their needs were met. (As of this writing, these features are being designed for the next release of REQADM, which should be finished by the time this paper is actually published and presented.)

New Queuing Theory

While we were designing our new request system, there were a number of new ideas proposed and considered. One of the more interesting ones involved changing our fundamental queue-management philosophy. The hypothesis was that our "multiple queue, multiple server" system was inefficient, since every "server" (i.e., every EC support technician) had his or her own queue of tickets assigned to be worked on. In some cases, some tickets in a system administrator's personal queue would wait for other tasks to be completed, when another individual might have been able to take them on in the mean time. We can liken this to a grocery store check-out line, where you have to pick a line and wait in it, although a customer ahead of you may take longer than you expect, but you can't just jump over to a faster line without going to the back of the line again.

The new system proposed was a "single queue, multiple server" system. In this system, which we can liken to a bank teller queue, all the incoming tickets would wait in a single queue until a support person was actually ready to work on it. The theory was that tickets would move faster through the system.

We implemented a script called *getrequest*, which support people would use to get the next available ticket from the incoming queue, and assign it to themselves. They would then work on that ticket until it was finished, and run *getrequest* again.

There were, however, a number of concerns we had about basing our entire ticket-handling methodology on this new concept.

First of all, there will always be a number of tickets which must wait for some external event, such as a part arriving from a vendor, or a customer response, and while that ticket is "on hold," the next ticket on the list is grabbed and started. Before long, individual support members would build up their own queues of tickets again anyway, and eventually, we reasoned, they would stop taking incoming tickets while they tried to clear their personal ticket backlog.

Secondly, we have to recognize that systems administrators are human beings. We wanted to be very careful not to establish a ticket-handling system which relied too much on diligence to manually get as many new tickets as one could get. We were concerned that there would be too much temptation to delay grabbing a request if a difficult or unpleasant request is next in line.

Thirdly, along the same lines of human nature issues, there are a lot of other tasks systems administrators need to concern themselves with every day, so we feared that a natural inclination would be not to get tickets as often as possible, on the basis that we each know we are already quite busy with other important tasks and don't have free time to take on a ticket yet. In our original system, where tickets were dispatched to support people directly, there was more of a tendency to just accept the workload and work all the tickets in to the day somehow, so more work was actually accomplished.

However, there were still some interesting uses for the techniques we discussed for this queuing theory, and this did affect how we assign requests into certain "rotation" groups, as we'll discuss below.

Improved Queue Displays

An artifact of the old system was that a lot of ASCII files needed to be opened, parsed and scanned

in order to view what was in the incoming queue of tickets (for the ticket dispatcher), as well as an individual's personal queue.

Originally, this was done by a simple shell script which everyone would run every few minutes, rescanning all files every time. An improved version of this script was implemented which ran continuously in an xterm window. It would make a pass through the queue directories every few minutes, but would only parse and scan files modified since the last scan. Then the results were displayed in a sorted, color-coded table, helping a support person locate information quickly and easily.

Automatic Request Dispatching

To relieve the heavy burden of assigning tickets to suitable people without overloading them, an automatic system called "REQMGR" was created.

General areas of issue ownership were identified, and fake usernames created for them with names like "postmaster" (for mail issues), "operator" (for backups and restores), "unixreq" (for general UNIX issues), "ntreq" (for general Windows NT issues), etc. These owner names were called "rotations", since at any given time, some subset of our support group would be "on duty" for their shift on one of those areas as part of a rotating schedule.

The REQMGR script runs every five minutes out of *cron*(8) and scans the active queues for any tickets which the dispatcher assigned to a "rotation". For each of those, REQMGR looks at the rotation schedules to see who the current candidates are, and assigns the ticket to the candidate who is the least loaded with other tickets.

This has proven to be a great asset, and has been integrated into the new REQADM system still in use today.

Looking in Other Directions

At this point we were facing the prospect of having to "reinvent the wheel" by redesigning our entire ticket system. Before going to those lengths, we tried to investigate other products which might already work for us.

Remedy AR System

One option was the Action Request System by Remedy. This was (and still is) a tool being successfully used at other Intel sites, and was worth looking at for our purposes. We set up a Remedy server, dedicated a full-time person to configure and administrate it, and tried to move our staff and customers over to the AR System.

While AR System is itself a fine product and many sites are happy using it, we found that it didn't quite fit into the established way we preferred to handle tickets.

From our point of view, we needed a system we could customize heavily to our department's needs and preferences. Further, any deviation in the general approach to tracking tickets from the old system to the new one would be a significant stumbling block. Our customers rightly argued that as they were on tight development deadlines, a substantial change to the process by which they entered their tickets, and EC's process for resolving them, would impact their overall productivity and might slip project schedules.

One of the main features our customers absolutely required was the ability to send free-form text messages via E-mail to a "sys-help" alias, and have that message automatically enter the ticket system database. They wanted updates to be mailed to them every time an action was taken on their ticket. This further justified a different solution where we could customize all these interfaces to our liking. (AR System does allow mailed requests, but they need to adhere to a fairly specific format.)

Helpdesk Institute Conference

We also sent a contingent of our support staff to a conference sponsored by the Help Desk Institute [5]. They attended talks and workshops specifically oriented to running successful helpdesk centers.

They returned with a number of interesting new perspectives on help desk center management and structure, which influenced the redesign of our "Computing Support Center". They also returned with a report that the conference attendees who were the most satisfied with their request ticket management software were the ones who designed their own custom systems.

This helped us to justify going ahead with the design and implementation of our own custom system.

This sentiment has been echoed by many others in the industry, including D. Johnson of Network Computing Group, who stated, after describing a model for "ideal" trouble ticket systems, "It is hard to imagine that this whole system could come out of a shrink-wrapped box..." While mentioning that commercial RDBMS packages' screen form and report generation facilities may be a good start, he found it "difficult to integrate full trouble ticket functionality through these systems [6]."

Design of a New System

We assembled a team, composed of technical contributors and management from the EC group, to work out the design of the new system. Over a period of several weeks of brainstorming, discussion, and only occasional arguments, the following system was sketched out, and development began.

In all of this, our goal was to design a trouble ticket system suited specifically to our needs in Engineering Computing, optimized to our specific support and business models.

It came as a pleasant surprise when we later found RFC 1297, *NOC Internal Integrated Trouble*

Ticket System Functional Specification Wish-list [6] and compared it against our REQADM system. The system we developed independently matches almost the entire wish-list in the RFC. To some degree, this gave us a feeling of validation that we must at least generally be on the right track with REQADM. It also validates the RFC, in that it accurately portrays a large support organization's ticket tracking needs.

New Computing Support Structure

Part of our efforts to revamp our customer support organization involved changing the way we ran the support center.

Originally, our support model was a "free-for-all" affair, where all tickets submitted by customers were directly assigned to an appropriate systems administrator. This was frustrating to EC and our customers alike, since it tended to overload the sysadmins who already had full workloads of their own, in addition to carrying a backlog of one or two dozen tickets (which just had to be "worked in" to the day somehow).

We looked at the way computing centers were being run elsewhere in our company, took the best of the ideas we found from each group, and created a system that would work well for our specific needs.

We created a "Monitoring and Control Center" (or "MCC") which would use automated network and host monitoring tools to proactively watch for trouble, fixing problems *before* customers started complaining about them.

The name of the center was later changed to the Computing Support Center ("CSC"), because some of our customers thought the "monitoring" and "control" words referred to our customers rather than their systems, and thought it sounded too Orwellian for their comfort.

To relieve much of the burden of ticket handling from the bulk of our systems administration personnel, we staffed the CSC with a full-time crew to be our "front line" tier, answering phones and handling all the tickets they could. This immediately removed all tickets not requiring heavily specialized expertise from the majority of sysadmins. Eventually, the CSC was able to handle more and more kinds of tickets directly, as area owners were able to train the front line to handle common problems and procedures such as new account creation and mailing list administration.

To supplement the front line crew, everyone else in EC (up to and including senior management) agreed to serve a week-long "rotation" in the CSC. During this week, they do nothing at all except handle request tickets that come in, for which the front line people aren't trained to handle, such as specific technical areas of UNIX or NT administration. By having them in the same physical room (the CSC), everyone on duty that week maintains a high level of communication about the current issues being dealt with. As an additional benefit, the full-time front line people (who are typically our newer, less-experienced administrators) get hands-on experience watching and helping the more seasoned administrators handle these issues. Many of the front-line crew eventually get moved on to specific systems administration positions based on their experience gained in the CSC.

To offset the stress and turmoil of dealing 100% with user requests and telephone calls for a solid week, once a sysadmin's rotation week is completed, they get to leave any unresolved tickets behind for the next week's crew to continue working on, and they don't have to return for ten more weeks. So the next ten weeks can be as close to "uninterrupted" as possible for any systems administrator,[1] and more work on projects and other administrative work can be accomplished than was possible when everyone was carrying that burden all the time.

Life Cycle of a Request Ticket

Requests for work are initiated in a variety of ways. Most commonly, a customer will have a question or will notice that there is something wrong in their work environment. They will use one of the interfaces described below to communicate their issue, directly entering a new ticket into the REQADM database.

Alternatively, they may call our CSC hotline and explain their problem to a front-line technician. Unless it's a trivial question which can just be answered in a couple of sentences, the front-line tech will call up a blank ticket form on their screen and enter a request on behalf of the customer.

Sometimes, however, one of our myriad system-monitoring scripts discovers a problem and either solves it directly or generates a REQADM ticket for a system administrator to look at.

Once the ticket enters the REQADM system, by whatever means, it has a "new" status, which means that it has newly arrived and has not yet been assigned to a system administrator.

Birth of a Request Ticket

Ideally, we want to prompt the user for specific information when they submit a ticket into the system. This has taken a fair bit of work, since in our old system customers would simply E-mail a free-form text message to a designated mail address. The problem with such unstructured input is evident when you receive request tickets which look like:

```
From: j_random@ichips.intel.com
To: sys-help@ichips.intel.com
Subject: FIX IMMEDIATELY

My environment is broken.
Pls fix now.
```

[1] In other words, not much, but it's definitely an improvement.

Naturally, immediately after submitting the ticket, the user leaves the office for the day, leaving the EC support group at a loss to even understand the complaint.

From the customer's point of view, however, the ease and simplicity of sending a simple mail message to initiate a ticket is very attractive, so our new system had to continue to support this.

However, the new system includes interfaces which we advertise as the "preferred" method for contacting us. The most basic is to simply type *request* at a shell prompt. This brings up a data-entry tool as shown in Figure 1.

To avoid overwhelming the user with too many input fields, the entry form is tabbed "notebook" style. The first tab, which identifies the user and provides us with contact information, is totally pre-populated by looking the user up in our corporate employee database. Most users need only check this page for accuracy and move on.

The second tab identifies the specific system being complained about. This section is only needed for workstation-related tickets and can be ignored when not applicable.

The next tab allows us to specify the attributes for this ticket. The request queue is one way we can organize types of tickets. The vast majority of tickets submitted go in the default "request" queue. We have defined other queues for special purpose situations, such as requests from EC to outside vendors, long-term projects, TO-DO lists, and so forth. Each queue has its own rules for agreed-upon resolution timeframes, and can be separately queried for reports.

The other two fields are of the greatest importance to the customer submitting the ticket. In the old system, there was but one way to indicate priority. You could mail your request to 'sys-help' or to 'hot-help.' Having only one bit of resolution for ticket priority proved woefully inadequate. For example, there was no way to distinguish "this task is extremely urgent and the world will end if it's not done in two hours" from "this task can wait until Friday but is extremely important that it really be completed at that point, or the world will still end."

To solve this, we prompt for two separate data points. The first is a four-level "importance" rating:

Urgent: Emergency situation; multiple users are down; all work stopped.

High: Significant impact to work being done; resolution needed ASAP.

Medium: Default priority – some impact to work; other work may still progress normally; resolution needed when reasonably possible.

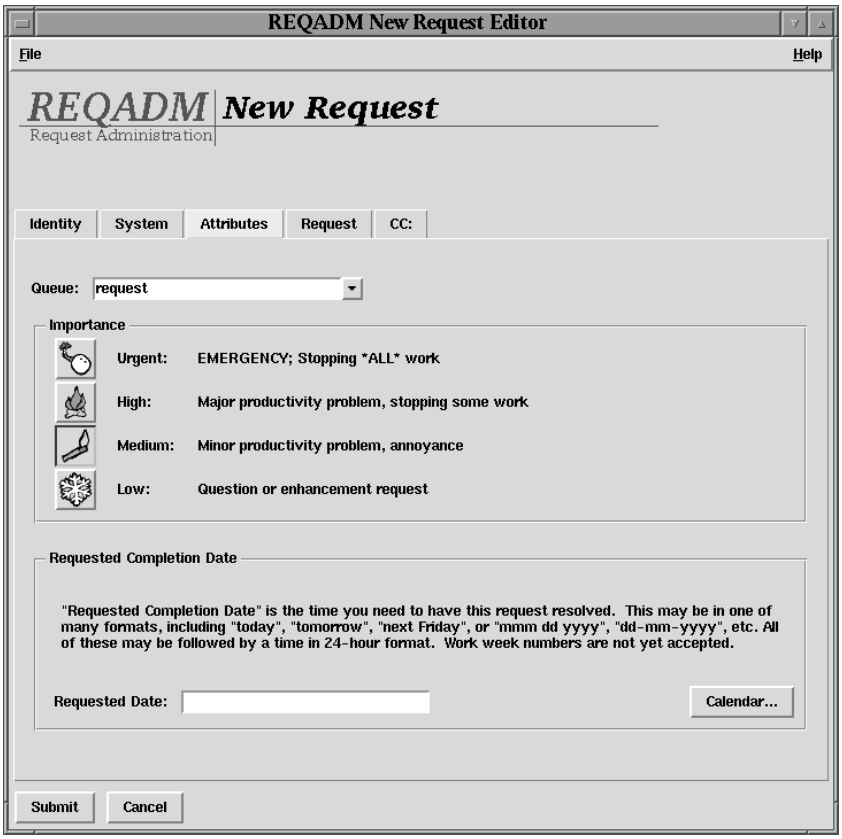

Figure 1: Request data-entry tool (''attributes'' tab).

Low: Not very important; request for routine enhancement; random questions.

This indicates how much the issue is impacting the customer – how important it is to be resolved – without explicit regard to a deadline for completion.

The second data point is a requested due date. This indicates an explicit point in time when the customer requires completion of the request. If none is specified, a default deadline is calculated based on importance rating and our contractual SLA with the customers. (If the customer requests a deadline earlier than our SLA contract has agreed to provide, it's not considered binding, but it's still useful to know.)

With these two data points, we have a much more accurate indication of how best to prioritize this ticket in the list of other tickets we're dealing with.

The next tab allows the customer to enter free-form text describing the problem in detail (see Figure 2). To help inspire the customer to specifically mention how this problem might be reproduced in the lab, we explicitly ask for this on the form.

The input tool can be customized to add special fields to this tab as appropriate for specific problem types. For example, if a customer is requesting additional disk space, it may add fields asking how much space is needed, on what fileserver, for what group, etc. Or, if access to a UNIX group is requested, it could look up the groups the user currently belongs to and set up a menu of other groups to select from.

The last tab allows us to specify a list of other people to be copied on all mail correspondence regarding this ticket.

Once we are finished entering the relevant data, getting the ticket into the system is only a click away. REQADM gives us immediate confirmation with the ticket number for our future reference.

Assignment

Meanwhile, back in the CSC, a front-line technician is on duty to watch for, and dispatch, incoming tickets. On his REQADM display, new tickets appear highlighted at the top of the queue list; see Figure 3.

These tickets can be opened directly from this list, and examined or edited by the support technician; see Figure 4.

From the type of problem in the complaint, the dispatcher assigns the ticket to another support technician. For example, if the request were about a problem with a mailing list, it might be assigned to "unixreq". This is the standard rotation for UNIX requests. This will then automatically be assigned to the next

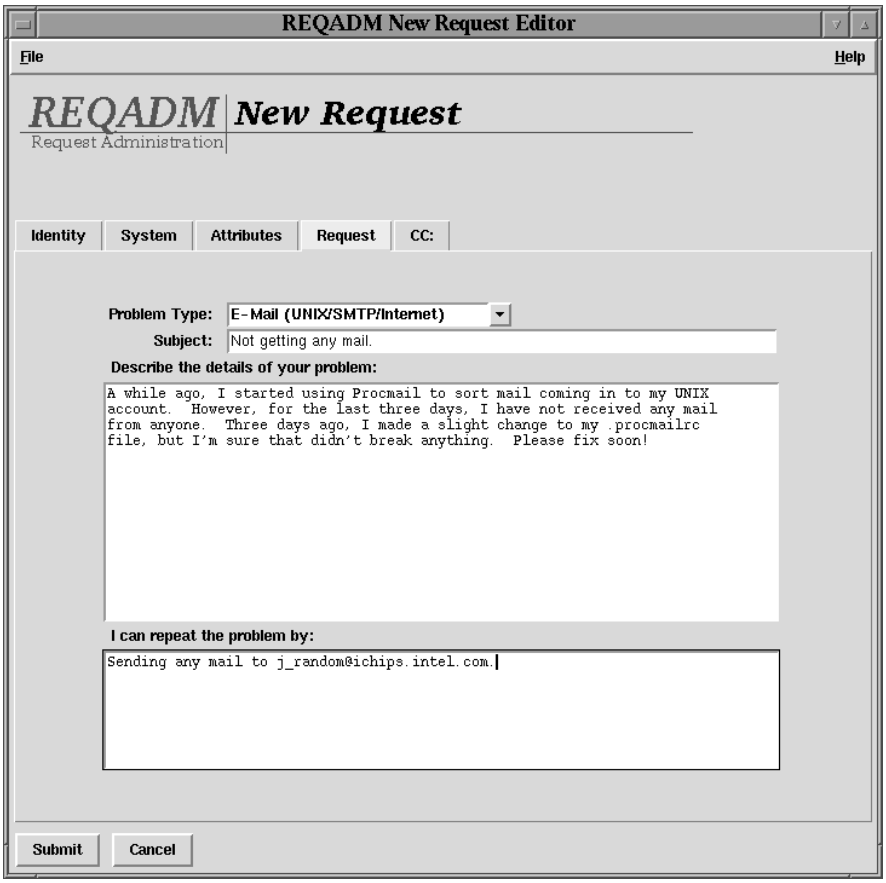

Figure 2: Request data-entry tool (''request'' tab).

available UNIX engineer in the CSC, who will look at it, and hopefully be the one to actually own the ticket through to completion.

However, in some cases, the ticket must be passed up the line to an engineer who owns a specific area of the environment. For example, if the mailing list was beyond the capability of the CSC to handle (perhaps a bug in Majordomo, or a Sendmail anomaly), it would be re-assigned to "postmaster", which the system will automatically send to the current Postmaster-on-duty for the day. It could also have been assigned directly to a specific technician.

The person or group to whom the ticket is assigned is known as the "ticket owner" and is responsible for seeing that it gets resolved.

Here's where the different queueing models discussed above come into play. Since some rotations, like "postmaster", are staffed with specific people dedicated to own specific areas of the environment, the traditional "multiple queue, multiple server" model is an optimal way to supply waiting tickets to the specific area owners on duty.

For distribution of the bulk of our requests, however, which are handled by those working in the CSC, the newer "single queue" model is actually more effective. This blending of philosophies has proven quite successful for us.

Priority

Each ticket has a priority, which is on the same four-level scale as the "importance" rating set by the

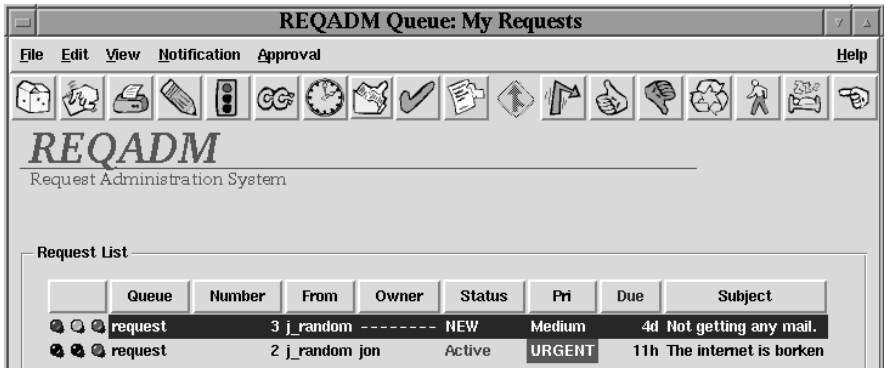

Figure 3: Queue display, showing new ticket arrival.

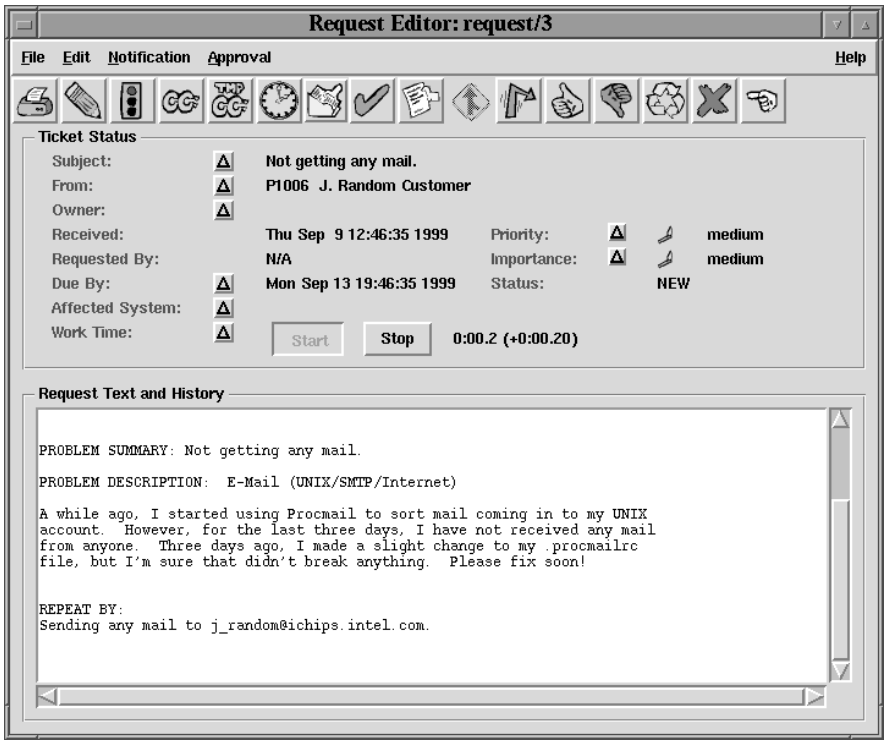

Figure 4: Examining a ticket.

customer. However, the importance rating is just an indicator from the customer. The true queue priority is set by the systems administrator. Generally, this will follow the customer's importance rating, unless EC disagrees with the customer as to the actual urgency of the problem.

When the systems administrator views her queue of tickets she owns, this list is sorted (by default) according to the amount of time remaining before it's due, which is a combination of the committed due date, priority and contractual SLA expectations.

Status Updates

As the ticket owner works on trying to solve the request, she will make updates, either as notes to herself and other support engineers, or to the customer, giving progress updates, asking questions, and recording what work was actually done.

These updates are made within the REQADM system, and are recorded in the permanent history of the request ticket. The results of each ticket editing session (which may include multiple actions such as updating the committed completion date, changing priority and adding a status note) is summarized by E-mail to the ticket submitter, owner, and other interested parties specified in the ticket's "CC:" list.

Checklists

Depending on the problem type being reported, the system may also attach pre-defined checklists of tasks which need to be followed to resolve that kind of issue. This assists the support engineer, especially if they are trying to perform an unfamiliar task during their CSC rotation week.

The items on the checklist must be completed before the ticket can be marked as resolved.

Holds

From the moment a request ticket enters our system, a "clock" starts ticking down the hours until it comes due according to our negotiated SLA with our customers. There will be times, however, when we are waiting for the customer to get back in touch with us before we can proceed any further.

In these cases, we place the ticket "on hold" which makes it clear to anyone scanning the queues that EC is not actively working on the ticket until some external event (usually communication from the customer) takes place. It also stops the "clock", effectively extending the due date by the length of time the ticket was on hold.

The next action logged to the ticket automatically removes the hold and returns it to active status.

Resolution

Finally, the customer's complaint is solved, and the support engineer is breathing a sigh of relief and feeling the satisfaction of a job well done. The ticket is marked as "resolved" and again the SLA clock stops counting against us (since we've completed our assigned task).

Categorization

At any point up to resolution, the ticket may be categorized. We have defined a multi-level menu hierarchy for specifying the root cause of the problems we see. By tagging each ticket with its root cause category, we can generate many interesting reports and analyze what parts of our environment might need extra attention.

For our example mailing list ticket, the engineer resolving the ticket would select the "mail" category, and the "mailing list problem" sub-category.

Supervisor Approval

In some rare cases, such as when the customer escalates their issue to EC management due to a disagreement or problem with the ticket owner, an EC manager must approve the resolution of the ticket. Normally, this is not necessary.

Customer Approval

This is where things get interesting. One of the problems we had with our old system was the perception among some customers that EC personnel were marking tickets as "closed" without fully solving the problem at hand. We had to keep track of which customers wanted us to never close a ticket without explicitly checking for their satisfaction first, and those who were annoyed if we bothered them to get that approval.

With the new system, we automated a customer approval process, which has made both sides happy. In this system, when EC thinks the ticket has been adequately completed, it's put into a new "resolved" state. The ticket is definitely no longer being worked on, but it's not really quite dead yet, either. It exists in a nether region or "ticket purgatory" for a while, and keeps its "resolved" status until approved by the customer.

At this point, the customer is sent e-mail informing them that the ticket has been resolved, and asking them to confirm this by approving the closure of the ticket. They may do so as simply as typing an X in the correct box and mailing the E-mail message back where it came from. Or, they may access the ticket using an interactive tool and click on the "approve" button.

Once this is done, the ticket becomes fully "closed" and is no longer a concern.

On the other hand, if the customer is still not satisfied, they may indicate that they deny the approval, and the ticket is automatically placed back to active status.

If the customer gets busy and doesn't bother explicitly approving closure of a ticket, the REQADM system will automatically close it after a grace period (currently five days) expires.

Resurrection

If a ticket has been fully closed for a while and the problem arises again, a customer or administrator

can search for it and reopen the ticket to resume active work once again.

Interfaces

One of the most important requirements of the request system is that it be easily accessed from just about any kind of environment.

Our customers were already used to submitting tickets via E-mail, so at a minimum that practice needed to remain possible. The challenge to that, of course, was to find a way to allow people to include ticket attributes such as importance and due date (q.v.), but still allow free-form text for everything else, and accept mail even if it doesn't contain special fields.

The solution we found was to have the mail receiver accept a "template" which provides special fields for all those attributes. If a random-text message is submitted without the template, a warning is sent back to the user advising them that they would get better results by including the template next time, allowing them to specify all the attributes appropriate to their issue.

Our preferred method of submitting tickets, however, is to use the GUI tool described above. For times when a windowing system is unavailable or undesirable, the *request* program will switch to ASCII mode and run the user through the ticket creation process, prompting for each line in series.

Likewise, ASCII versions of all the REQADM toolset are available when a GUI isn't an option.

To help facilitate automation of ticket submission and administration, all functions of REQADM may be performed using batch-mode commands, without requiring any user interaction at all.

There are also API libraries for writing Perl or C++ clients to the REQADM system, to accommodate any level of future interface desired.

To help remote users who wish to access the ticket system, or for those who simply prefer such things, there is also a fully-functional web interface for submitting and manipulating tickets in the system.

Personal Options

In a customer base as large as ours, there is no shortage of opinions for how the system should behave, what mail notices should look like, and every other aspect of the system.

Rather than try to compromise between everyone's desires, and recognizing that different people mentally organize data differently (and would benefit from different data presentation formats), we provide some personal configuration options. Every user has a "profile" maintained by REQADM which determines how they see mail formatted, whether to page them when tickets are assigned, and a large number of other factors.

E-Mail Correspondence

When anything happens to a ticket, mail is sent to the ticket owner, submitter, and everyone on the ticket's CC: list. Each person's copy is separately composed and mailed, so that the recipient's personal formatting preferences can be applied to their individual copy.

The mail body always begins with the comments placed by the person editing the ticket, since those will most likely be the most important text to consider. Following that is a summary of the actions taken and the current status of the ticket.

The mail headers are created in such a way that if anyone replies to the mail directly, it is sent back to REQADM itself. This reply is added to the ticket history and re-mailed to everyone involved in the request. In this way, we can exchange information with a customer or other party, and as far as the customer is concerned, it's just a simple E-mail conversation.

Nobody has to invoke a special tool to update the ticket along the way; REQADM just tracks the conversation automatically in the ticket history.

Technical Design

The REQADM system centers on a single server daemon (*reqadmd*). This server maintains its own database containing request tickets and personnel information.

Databases

The server contains its own lightweight embedded database engine[2] to rapidly organize and access request ticket information. The personnel database is fed from the following sources:

1. The UNIX password file, which provides a mapping of user ID number to username. This is helpful for REQADM clients to identify who is running them, and grant an appropriate level of access for each user, without requiring users to "log in" to REQADM with yet another password.
2. The "usertable" file. This is a simple ASCII file mapping username to employee ID number. REQADM actually uses employee ID numbers to identify users, since that needs to be a key which will never (or at least very rarely) change. Experience has shown that OS usernames and user ID numbers tend to change too often.
3. Corporate personnel file. This is a simple ASCII file which provides information about each person, including phone numbers, mailstop, office location, and employee ID number. This allows REQADM to help EC staff find customers when they need to follow up on a ticket.

Every night, REQADM checks these sources for any information that needs to be updated. (For

[2]It actually uses the Athena DBMS library from Software Alchemy, Inc.

example, if a person moved to a new office, the corporate personnel file will show their new location, and REQADM will pull that change into its copy of the database too.)

REQADM can also be configured to access some of this information on the fly to allow new people to use it even if they are not currently in its personnel database. Specifically, it'll try contacting:

1. A local LDAP server we have which lists all known E-mail addresses for all Intel employees (many people have accounts at multiple sites and may have mail addresses at each, as well as the overall "intel.com" domain). This will help REQADM resolve unknown (to REQADM) mail addresses to employee ID numbers, which it can look up further.
2. A local server providing the above-mentioned corporate personnel information. REQADM will use this to create personnel records in its database for new people when first encountered.

Whichever combinations of the above files and services reasonably apply to other sites' environments can easily be set up to support REQADM.

Additionally, the *reqadmd* server accesses the tickets stored in human-readable MH-format ASCII files (in pre-2.0 versions; after 2.0, REQADM stores all tickets in the database but writes out ASCII recovery files in quickly-parsable machine-friendly ASCII formats, Just In Case. Normally, though, those files are written but never read by REQADM 2.*x*). The server is the only process allowed to directly touch either the database or ASCII files; as such, it arbitrates security and locking amongst the request system users.

Clients

Everything else the sysadmins and customers encounter is a client program or script which contacts the *reqadmd* server via TCP/IP sockets, requests it to perform queries or modifications of tickets, and reports back to the user.

Clients exist for submitting tickets, setting personal look-and-feel preferences, browsing the queues and editing request information.

Most of the clients respond to whatever environment they sense they are being run from (GUI, ASCII, WWW, or batch) and interact with the user appropriately.

Supporting Scripts

A number of scripts support the operation of REQADM. The most interesting of them are listed below:

REQMGR manages automatic assignment of tickets to sysadmins who take turns being "on duty" for specific tasks such as postmaster, webmaster, and account creation. It checks schedule files to see who the candidates are to receive each ticket, and then checks the REQADM server to see who has the smallest active request load.

Reqadm-mail-receiver takes all the incoming E-mail (whether new incoming tickets or mailed responses to existing tickets) and invokes the other clients necessary to get the mailed information into REQADM.

Reqadm-auto-close runs as part of the nightly maintenance procedures, finding all tickets which have been marked as "resolved" but have not been touched for a designated number of days. These are considered to be abandoned by the customer, who has not responded to approve or deny the resolution. Those tickets are then marked "approved".

Extension APIs

In order to allow metrics-gathering scripts and other new clients to be written as desired, a comprehensive C++ API library is provided allowing virtually any new facility to be added to the system.

A reasonable subset of those functions are available in a Perl module, allowing scripts to access the REQADM server. We make significant use of this module for our statistical analysis scripts.

Implementation Notes

The core of REQADM (server, clients, libraries) consists of nearly 42,000 lines of code (mostly C++, with a bit of Tcl/Tk and C as necessary). This is supplemented by about a dozen shell and Perl scripts responsible for the ongoing maintenance tasks.

It has been built successfully on Solaris, AIX, HP/UX, SunOS, Linux, FreeBSD and Windows NT. (Currently the server must be SPARC Solaris or FreeBSD/i386.)

Phase I: Gradual Integration

The REQADM program took almost a year to go from initial design committee meetings to the completed version 1.0 release. At that point (early 1998), we were ready to cut over from the old MH system to the new REQADM server.

We needed to be extremely cautious, however, since the request ticket system is so critical to our department. In case of trouble, we needed a fast escape route to switch the old system back online.

The major consideration to that end was the decision to keep the old MH-style file format from the old system. REQADM was designed to read and write the old-style files, and maintain an "overview" database tracking the status and ownership of the tickets in the discrete disk files. This allowed for fast database queries and reports of this overview data, but if the actual contents of a ticket needed to be examined, the disk file had to be opened and its text format parsed.

In this way, if REQADM failed in some major way, we could simply turn off the REQADM server and continue using our old scripts on the same data files, with no loss of information or stoppage of work.

We began with an "alpha test" phase internally within the EC group. We started a new REQADM server without an existing set of tickets, and ran

scripts to randomly generate a few hundred request tickets. These tickets contained instructions for the owner to perform 6-8 randomly chosen tasks within the request system, such as "Assign this ticket to *(random other person)*", or "Change the priority to 'urgent'". We used these tickets to further test the REQADM system, as well as to train our support staff on the new interface they'd be using.

Once we were satisfied with that, we cleared out the random tickets and asked our support staff to use REQADM for all internally-generated tickets. We also asked a select group of customers to try using REQADM for all their new issues as well. This gave us more "real world" testing of REQADM, while allowing us to still use the old system with most of our existing customers.

To make this work, we hacked REQADM to start issuing new request tickets at ticket number 70,000, which was a significant gap from the largest ticket number in the MH system (which was still under 60,000 at the time).

When we believed REQADM was ready for production use, we ran a Perl script to import all the MH-style tickets into the REQADM database (adding them to the higher-numbered ones which were already there from the customer testing period), and told the customers that REQADM was the way to submit tickets from then on.

We also provided them with a step-by-step tutorial for each of the customer interfaces to REQADM. This tutorial allows them to submit sample tickets to an "autotest" queue, which is ignored by humans, but monitored by a cron script which will pretend to be a sysadmin working on their ticket, doing random things to it and eventually closing the ticket. This lets them figure out how the system works without unduly annoying any of the EC staff.

REQADM was a success and the emergency back-out plan never had to be executed.

Phase II: Taking the Plunge

As this paper is being written, we are beginning implementation of REQADM version 2.0, which will include quite a number of improvements to the system described here. By the time you read this paper, 2.0 should be completed and ready for release.

One of the major changes will be to abandon the old MH format, and to store all ticket information completely inside REQADM's database. This will allow us to search for tickets by text content as well as attribute and subject information, and will improve the speed of handling tickets.

Once we do this, we won't have the "safety net" of falling back to the MH scripts, but since using REQADM for over a year and a half, we don't believe that will be an issue any longer. The 2.0 system will have its own internal recovery systems to help prevent loss of data in the event of database corruption or other problem.

Success of REQADM

Our customers and support staff have been very happy about the improvements REQADM has provided to our ticket handling process in the PMD EC. We've been gratified to have been asked by other Intel groups to give them copies of REQADM to use for their own ticket management.

According to programming folklore, one of the marks of success for a software system is when someone successfully applies it to a problem which the programmer never anticipated. This happened for REQADM when one of our product testing groups decided to adopt REQADM to track downtimes for their chip- and board-tester systems, and for keeping maintenance information with the vendors who supply those systems. By keeping the REQADM tickets' timestamps updated accurately, and by asking for just the right metrics from the REQADM server, they now are able to get automatic mean-time-between-failure and mean-time-to-fix statistics for their test equipment, in addition to the normal usage of REQADM to track active issues with their staff and vendors.

Fringe Benefits

The EC group has also been able to leverage REQADM to do more for us than just track requests for help from our customers.

For example, we created a queue of requests the EC group places with outside vendors. Now when we call (for example) Auspex to get a new fileserver part, we can create a ticket and assign it to our Auspex sales or service representative. The vendor can use the mail interface to REQADM to add updates and other information to the request ticket along the way, giving us a permanent record of service for our systems. When we pull a service ticket for viewing in REQADM, we can ask for all other tickets filed against the same system, and see trends that might need to be discussed with the vendor.

Metrics Gathering

One of the most important topics for EC managers to consider is how to keep track of the general issues the EC group is facing, and how to reduce the number of backlogged tickets (by getting our teams to work more efficiently), and to reduce the number of tickets coming *in* to the system (on the theory that if we proactively improve the quality of system services, the customers won't have so much to complain about).

In order to do this, they need real data to work from. We've been gathering various statistics on weekly numbers of request tickets opened, closed, backlogged, late, and so forth, which get translated to various graphs and charts for managers to look at and discuss in their meetings.

The system also tracks the "categories" assigned to each ticket by the ticket owner. These categories show the root cause of the problem which prompted the ticket to have been entered in the first place.

The categories are organized into multiple tiers. At the outermost level we might have a list of categories like:

- Account Administration
- Fileserver/Disk Management
- E-Mail
- Network/Internet
- Performance Issues
- Printing/Plotting
- User Environment

Under each of those are sub-categories and sub-sub-categories. For example, under "E-Mail" we might find:

- Bounced, Pilot Error
- MTA Problem
- MUA Problem
- Bogus .forward File
- Spam/Abusive Mail

We generate reports at the top level each week ("How many mail requests did we get? How many network problems?"). We also developed an interactive Request Analysis Tool ("RAT") on the web which allows one to browse these statistics, opening up a top-level category to see *why* there were so many tickets of a particular type. From these statistics, we can recognize problems in the environment early enough to reduce their impact.

Multi-Site Use

Our support group is divided between geographical locations, with some of us in Oregon and others in Washington. While, for the most part, the Oregon and Washington teams don't tend to work together on individual tickets, there is still a need to have our request systems accessible to each other.

We have had reasonable success having those at remote sites run clients which connect to our server, giving them access to our request tickets. However, this doesn't work well when the wide-area network between Oregon and Washington goes down.

We're developing a new protocol which will allow two or more REQADM servers to communicate between themselves, passing information occasionally between them. This will allow our Washington team to maintain their own server and local request queues. When they get a ticket that belongs to the Oregon team, they can just click on a "cross-site transfer" button and the server will move the ticket to the Oregon server. This feature will also allow people at either site to browse (but not necessarily modify) queues at remote sites.

Future Development

There is always room to improve any system, and we'll probably never be 100% finished with REQADM, as long as people keep asking for new features.

One thing we've considered is a tie-in to some kind of expert system or knowledge base. We'll probably do this in two places. First, when the customer is submitting a ticket, it would be helpful to have the system refer them to information and/or URLs offering advice for the kind of problem being submitted. This may even help reduce the number of incoming tickets.

Second, on the support side, we would like to expand REQADM's capability to cross-reference past tickets to see what was done previously to a ticket, as well as accessing a knowledge base of information about all areas of our environment.

We're also working on setting alarms for certain kinds of events (e.g., to warn when a ticket is about to become due), setting time-outs for the "on hold" feature, etc.

The idea of replicated backup servers for REQADM is also appealing, and we may pursue that at some future point.

Glossary

CSC Computing Support Center. The physical location where front-line and second-tier personnel sit during their week-long rotation. Phone support is handled there, as well as walk-in traffic.

EC Engineering Computing. The overall team of systems administration staff for our engineering department of customers. Contrast with IT.

IT The corporate Information Technology group. The computing support organization for everyone else in the company not served by Engineering Computing.

MCC The Monitoring and Control Center. The old name for the CSC, before someone thought it sounded too Orwellian.

POD Affectionate term for the CSC; see [4].

queue A collection of request tickets. The separate queues help organize tickets into broad classifications.

rotation A "group" which may be assigned a ticket. Through some manual or automatic means, a ticket is then dispatched out of that rotation to the next available real person who is serving a duty rotation for that kind of issue.

SLA The Service Level Agreement between EC and our customers. This specifies the expected timeframes wherein tickets should

be resolved, and what services we are willing to support, in what manner.

References

1. David H. Crocker, "Standard for the Format of ARPA Internet Text Messages," *RFC 822*, 1982.
2. Rémy Evard, "Managing the Ever-Growing To-Do List," *Proceedings of the Eighth USENIX Systems Administration Conference (LISA VIII)*, USENIX, San Diego, CA, 1994.
3. Tinsley Galyean, Trent Hein, and Evi Nemeth, "Trouble-MH: A Work-Queue Management Package for a >3 Ring Circus," *Proceedings of the Fourth USENIX Large Installation Systems Administration Workshop (LISA IV)*, USENIX, 1990.
4. William Goldman, *The Princess Bride, 25th Anniversary Edition,* Ballatine, 1998.
5. Help Desk Institute, *http://www.HelpDeskInst.com*.
6. D. Johnson, "NOC Internal Integrated Trouble Ticket System; Functional Specification Wishlist," *RFC 1297*, 1992.
7. Bryan McDonald, "QMH: A Problem Tracking System," *Proceedings of the World Conference on System Administration and Security*, 1992.
8. Joe Rhett, "Request v3: A Modular, Extensible Task Tracking Tool," *Proceedings of the Twelfth Systems Administration Conference (LISA '98)*, p. 327, USENIX, Boston, MA, 1998.
9. Elizabeth D. Zwicky, "Getting More Work Out Of Work Tracking Systems," *Proceedings of the Eighth USENIX Systems Administration Conference (LISA VIII)*, USENIX, San Diego, CA, 1994.

Author Information

Steve Willoughby is a Senior Systems Programmer at Intel's Performance Microprocessor Division, where he has been for the last six years playing (at various times) Auspex Administrator, Postmaster, Security Czar, Perl Programming Teacher and maker of internal software applications.

He discovered Version 7 UNIX while in high school (ca. 1980) and, apart from brief forays into VMS in college and failed attempts to hide from a couple of other operating systems, he's been spending most waking hours since then tinkering on UNIX in one form or another, either writing software or administering systems.

He lives in the Portland, Oregon area with his wife, son and assorted small furry creatures. He keeps a vintage Altair 8800 as a pet. In his spare time, he tries to avoid running a MUD system (*http://www.rag.com*). He can be reached by E-mail at <steve@ichips.intel.com>.

APPENDIX

This appendix shows a comparison between REQADM and other popular free ticket tracking systems.

This information was taken from the documentation accompanying the following packages. Not all features were clearly and thoroughly documented in each case, so this table may not be 100% complete. Time did not allow for us to install and extensively use each package for an in-depth comparison.

REQADM Version 1.2, with anticipated features which should be completed when this paper goes to press.

RFC 1297 The mythical ticket tracking system described in the RFC is compared with the real features of the existing tools.

Gnats Version 3.2 from ftp://prep.mit.edu/pub/gnu/gnats/gnats-3.2.tar.gz

Netlog Version 2.4 from ftp://ftp.jvnc.net/pub/netlog-tt-2.4.tar.Z

PTS Version 1.1a2 from ftp://ftp.x.org/contrib/pts-1.1a2.tar.gz and http://www.halcyon.com/dean/pts/pts.html

Req Version 1.2.7 and tkreq version 1.10 from ftp://ftp.ccs.neu.edu/pub/sysadmin/*

RUST Version 1.0b6pl2 from ftp://ftp.eng.utah.edu/pub/sys-admin/rust/rust-1.0b6pl2.tar.gz

Request Version 3 from http://www.noc.isite.net/software/Request/ was not compared since the web site did not contain an active link to obtain a copy of the software, and did not have documentation on line to describe all features in depth. However, from what information we have, it appears to be comparable to the other systems shown.

Feature	REQADM	RFC 1297	Gnats	Netlog	PTS	Req	RUST
General Features							
Running history in ticket	yes	yes	yes	yes	yes	yes	yes
Auto-timeout for escalation	no	yes	no	no	yes	no	no
Feedback to net monitors	*	yes	no	no	no?	no?	no
Fixed-input field templates	yes	yes	yes	no	yes	no?	no
Free-form text fields	yes	yes	yes	yes	yes	yes	yes
Problem classes or queues	yes	yes	yes	no	yes	no	yes
Assisted entry for fields	yes	yes	yes	yes	yes	no?	no
Help screens in tools	yes	yes	no?	no	?	no?	no?
Pass tickets to other site	yes	yes	no	no	no	no	no
Integrated with expert sys	no	yes	no	no	no	no	no
API for ad hoc queries/rpts	yes	yes	no	no	yes	no	?
API for custom clients	yes	yes	no	no	yes	no	?
Refer to old closed tickets	yes	yes	no?	no	yes	yes?	yes?
Re-open a closed ticket	yes	no	no?	no	yes	yes	yes
Database used	custom	N/A	no	no	custom	no	no
Merge duplicate tickets	yes	no	no	no	no	yes	yes
Put ticket “on hold”	yes	yes	no	no	no	no	no
Support Rotations	yes	no	no	no	no	no	no
Auto-assign to Rotations	yes	no	no	no	no	no	no
Queue Displays							
Auto-updating ticket disp.	yes	yes	yes	no	no?	no	no
Select view by attributes	yes	yes	yes	no	no?	yes	yes
Sortable by owner/submitter	yes	yes	no?	no	no?	no	no
Sortable by ticket priority	yes	yes	no?	no	no?	no	no
Sortable by queue/number	yes	yes	no?	no	no?	no	no
Shows ticket status/age	yes	yes	yes	no	yes	yes	yes
Customizable displays	yes	no	no	no	no	no?	yes
Phonetic subject searches	yes	no	no	no	no	no	no
Security							
Admin functions restricted	yes	yes	yes	yes	yes	no?	yes
Per-field ACL/permissions	no	yes	no	no	no	no	no
Admin-eyes-only notes	yes	yes	no?	no	no	no	no
Customers can note tickets	any	*	no	no	no	any	any
Customer approval for close	yes	no	no	no	no	**	**
Admin approval for reopen	no	no	no	no	yes	no	no

Feature	REQADM	RFC 1297	Gnats	Netlog	PTS	Req	RUST
Ticket Attributes							
Customer priority levels	4	yes	3	no	no	3	3
Support priority levels	4	yes	3	no	no	no	no
Time/date submitted	yes	yes	yes	yes	yes	yes	yes
Time/date resolved	yes	yes	yes	yes	yes	yes	yes
Time/date last updated	yes	yes	no?	no?	no?	yes	yes
Location/system of user	yes	yes	no	no	no	no	no
Machine/network involved	yes	yes	no	no	no	no	no
One-line problem summary	yes	yes	yes	yes	yes	yes	yes
Next action to be taken	yes	yes	yes	yes	yes	yes	yes
Ticket owner (comp support)	yes	yes	yes	no	no?	yes	yes
Time spent working on prob	yes	yes	no	no	no	no	yes
Time ticket took to close	yes	yes	yes	yes	yes	yes	yes
Supervisor approval of work	no	yes	no	no	no	no	no
Escalation to non-EC or mgt	no	yes	no	no	no	no	no
SLA due-date for prob type	yes	yes	no	no	no	no	no
SLA due-date for priority	yes	yes	no	no	no	no	no
Ticket status levels	8	yes	5	2	3	4	3+
Committed (manual) due date	yes	no	no	no	no	yes	yes
Requested (manual) due date	yes	no	no	no	no	no	no
Reports							
Summarize by network/host	yes	yes	no	no	no	no	no
Summarize by root cause	yes	yes	no	no	?	no	no
Summarize by date	yes	yes	no	yes	?	yes	yes
Mean Time Between Failure	yes	yes	no	no	*	no	?
Mean Time to Repair (work)	yes	yes	no	no	no	no	?
Graphical chart output	yes	yes	no	no	*	no	no
Environment							
Runs w/native window system	yes	yes	yes	yes	yes	yes	yes
Mult. open ticket windows	yes	yes	yes	yes	?	no	no
Auto. tools open tickets	yes	yes	yes	yes	yes	yes	yes
Pulls employee info from db	yes	yes	no	no	no	no	no
Pulls host/net info from db	yes	yes	no	no	no	no	no
Query env. when ticket open	yes	yes	*	no	*	*	*
E-mail notification/updates	yes	yes	yes	no	yes	yes	yes
Mail traffic logged to tkt	yes	yes	no?	no	no?	yes	yes
Batch updates w/o using GUI	yes	yes	yes	no	no?	*	yes
Failover to backup CPU/db	no	yes	no	no	no	no	no
X GUI interface available	all	N/A	no?	no	all	all	all
NT GUI interface available	all	N/A	no?	no	no	no	no
Web interface available	all	N/A	no	no	all	no	no
E-mail interface available	sub	N/A	sub	no	no?	sub	sub
ASCII interface available	all	N/A	all	no	all	all	all
Curses interface available	no	N/A	no	yes	no	no	no
Batch cmd interface avail.	all	N/A	all	no	no	all	all
EMACS interface available	no	N/A	yes	no	no	yes	no?
Free-form text submissions	yes	N/A	no	yes	yes	yes	yes
Assisted submissions	yes	N/A	yes	yes	yes	no?	no?
Users' custom preferences	yes	no	no	no	no	no	no
Mailed new-ticket receipt	yes	yes	yes	no	no?	yes	yes

* Possible to add by writing a custom script in Perl or other language.

** Possible by using existing features of the tool, but those features may not have been designed specifically (or exclusively) for that purpose.

all All major features are available in this mode.

sub A subset of features are available in this mode (e.g., you can submit new tickets by mail, and can add commentary to them, but can't otherwise administrate them via mail alone).