USENIX Association

Proceedings of the LISA 2001 15th Systems Administration Conference

San Diego, California, USA
December 2–7, 2001

Defining the Role of Service Manager: Sanity Through Organizational Evolution

Mark D. Roth – University of Illinois at Urbana-Champaign

ABSTRACT

In large university environments, a centralized academic computing organization is often responsible for providing campus-wide computing services. These organizations usually understand the need for system administrators and software developers, but the people they hire for these roles will often wind up managing service software as a secondary responsibility. This approach often results in poor service quality, overworked employees, and high staff turnover.

In this paper, I present the role of service manager as I have helped define it in my organization and explain how it addresses quality of service issues and staffing shortages. I also detail the organizational changes that my department has implemented to create a group of service managers, relate the process of implementing those changes, and examine some of the pitfalls that were discovered in the process.

Through the changes described in this paper, we have improved our quality of service, dramatically reduced the frequency of late-night pages, quadrupled the number of systems we can run with a fixed-size staff, and greatly increased staff retention.

Introduction

The Computing and Communications Services Office (CCSO) is the centralized academic computing department at the University of Illinois at Urbana-Champaign (UIUC). When I first started at CCSO in 1998, there were two groups that were involved in running most of our production services.

The Unix Systems Team (UST) consisted of 3-4 Unix administrators who were primarily responsible for the Student/Staff Computing Cluster, a group of Unix machines that students and staff could log into remotely to read email, access news groups, store files, etc. Because these services were tied so closely to the operating system, the same group of people maintained both the machines and all of the custom software running on them.

The second group involved in running CCSO's production services was the Software Development Group (SDG). SDG was supposed to be responsible for developing software as needed for CCSO's production services, but the developer of a given service usually ended up running the service once it went into production.

There were a number of problems with this organizational approach:

1. The primary responsibilities of the existing groups were system management and software development, and the costs of these tasks were well understood. However, there was no single group whose primary responsibility was to plan, deploy, and manage production services, so the cost of these tasks were completely hidden. In particular, no one budgeted for the additional staff needed to run each service.
2. Because the existing groups were expected to manage services in addition to their primary responsibilities, the staff was extremely overworked and undervalued. This exacerbated the already difficult problem of hiring and retaining good IT people in academia.
3. Even if the existing staff had not been overworked, managing production services was not part of the core competency of either of the existing groups. For system managers, knowing how to manage Unix systems is different than knowing how to run an LDAP server or a news server. For developers, the need for blocks of uninterrupted time to develop software is inconsistent with the time demands of maintaining a production service.
4. Because most of our services were being run by the developers directly, there was very little separation of production and development. This made it extremely difficult to track changes and keep our services stable.

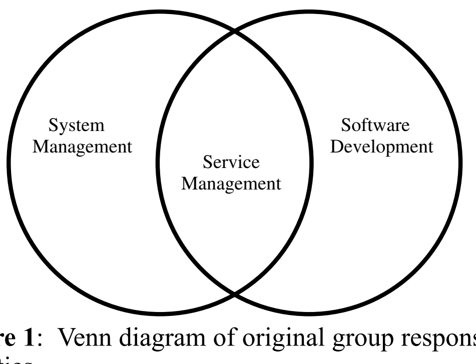

Figure 1: Venn diagram of original group responsibilities.

I first joined CCSO as a member of the Workstation Services Group (WSG), which provides support for Unix workstation owners in other departments on campus. One of WSG's most successful services is the contract administration program, in which Unix workstation owners pay WSG to manage their systems for them on a per-system basis. At the time, WSG was not involved in running CCSO's production systems, but we had built a reputation as a talented and effective group of Unix administrators, and management had taken note.

A New Approach

I first became involved in examining our method of managing production systems in mid-1999. WSG was approached about helping UST with some Student/Staff Cluster upgrades during one of their staffing shortages. That round of upgrades went very well, which led to discussions about how we could be of assistance on a more permanent basis. After reading papers like "Bootstrapping an Infrastructure" [1], I felt that we could improve things by forming a new Unix administration group based on long-term thinking, standardization, and scalability. The result was the formation of the Production Systems Group (PSG) as a subset of WSG.

(Another common way of addressing staffing shortages in academia is to make use of undergraduate labor, as described in "Guerrilla System Administration" [2]. It should be noted that we do employ a small number of undergraduates, but we cannot use them to directly manage our production systems, since they cannot be expected to be on call on a 24x7 basis.)

As PSG was getting off the ground, we quickly realized that we couldn't achieve the scalability we wanted without defining our relationship with our customers. As we started to set the boundaries of this interface, the need for a new service management group became apparent.

The Service Manager

Our customers on campus see the services we offer directly; they don't see system administration or software development as things that affect them. As a result, we needed to define a role whose primary responsibility was not systems or software, but services – in effect, this is the person who's looking out for the end-users' best interests, since they are his customers. Using a term mentioned in "Deconstructing User Requests" [3], we called this person the service manager.

An individual service manager is responsible for planning, implementing, and maintaining a given production service, including the following activities:

1. Initial Investigation
   - Work with management to develop a business plan and list of specific requirements for potential production services.
   - Evaluate software options that meet the requirements, including buying commercial software, using open source software, and developing new software locally.
   - Based on the software evaluation and on input from the system managers and the Operations Center, choose the best alternative for implementation (both hardware and software).
2. Production Deployment
   - Install and configure the service software on the hardware configured by the system managers.
   - Produce internal documentation of the production service configuration, including operational procedures such as backing up and restoring service-related data, adding and deleting users, etc.
   - Coordinate with the Documentation Group to produce end-user documentation for the production service. Also work with User Services to ensure that they're prepared to answer questions from users of the service.
   - Designate and train a backup service manager for the production service.
   - Coordinate monitoring activities for the production service with the Operations Center.
   - Coordinate the exact date and time that the service will go into production with the Operations Center and the system managers.
3. Ongoing Maintenance
   - Coordinate ongoing maintenance of the production service with the system managers and the Operations Center.
   - Coordinate with User Services and the Documentation Group to communicate service changes to the user community.
   - Respond to problem reports from User Services or the Operations Center in a timely fashion.
   - Maintain a list of desired enhancements to the service that can be addressed during the next development cycle.

As a direct consequence of this definition of the service manager's role, the new structure that we were targeting for our entire organization quickly became clear (see Figures 2 and 3).

Advantages

In early 2001, we got approval to form the new Production Applications Group (PAG) to take on the role of service manager. The group initially consisted of three positions, one of which was the manager of the group. Although the group is still in its infancy, we are already seeing several key benefits.

Economy of Scale in System Management

Because PAG is taking on the role of service manager, PSG has been able to achieve the scalability we were striving for. We've been focusing on building a scalable systems infrastructure, standardizing our procedures and system configuration process, and developing tools to automate as much of our work as possible. Because we spend time anticipating and avoiding problems instead of fighting fires, we've been able to provide a consistent, up-to-date environment and stay responsive to customer requests.

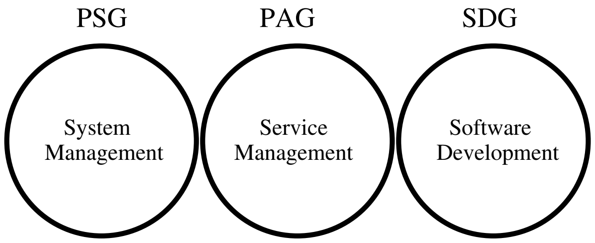

Figure 2: Venn Diagram of current group responsibilities.

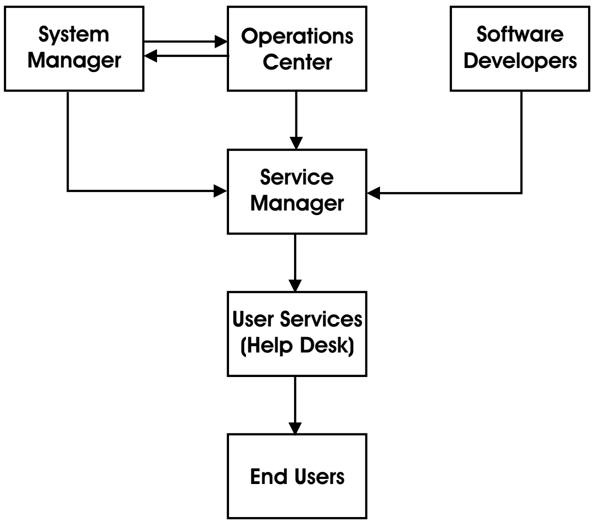

Figure 3: Desired CCSO organizational structure (arrows point from provider to customer).

Clarification of the Developer's Role

Just as PSG's existence helped define the role of the service manager, PAG's existence has served to clarify the role of the software developers in SDG. The software developer's role is to work with the service manager (who is his customer) to do any in-house software development necessary for the creation and integration of a service. The service manager provides a list of written requirements, and the developer is responsible for designing, implementing, and documenting the service software before handing it off to the service manager.

Because they are now able to focus on development, SDG is in the process of standardizing their development practices. They are beginning to think about revision control systems, code reviews, documentation, best common practices, and other development infrastructure standards.

Easing of Staffing Problems

Basic psychology proves that people are happiest when they have authority and responsibility in equal measure. The division of labor between system managers, service managers, and software developers has allowed us to define the scope of each group so that they each have authority and responsibility over their respective arenas and well-defined customers who provide feedback. These changes have increased job satisfaction across all three groups, which has helped tremendously with staff retention. Making our positions more desirable has also made it easier to hire new staff.

Improved Communication and Customer Focus

The process of creating PAG and clarifying the roles of PSG and SDG has allowed each group to focus on who their customer is. This improves our responsiveness to end-users, provides a higher quality of service, and helps ensure that everyone's needs are being provided for. The result is a higher overall quality of service: Services aren't disrupted very often, and they're fixed quickly when they do break.

Providing higher-quality service has also led to more positive feedback from both customers and management, which has been another boon to job satisfaction.

Implementation

The process of creating PSG and PAG has moved very quickly by organizational standards. It started in the summer of 1999 with the formation of PSG as a subset of WSG. For the next year and a half, PSG's focus was on developing our infrastructure and gaining acceptance. In the spring of 2001, as PAG was formed and began to develop its own infrastructure, PSG became a distinct group in its own right.

Getting Buy-In

Because the formation of PSG was worked out with the cooperation of management, we didn't have to worry about getting additional buy-in from that level. However, it was extremely important to get buy-in from the developers (who were managing our production services at the time), since they were to be the bulk of our customers. Even if we had been in an environment where management could force the developers to work with us, we needed to establish a cooperative relationship with them to keep everything working smoothly.

Our general approach to getting buy-in from the developers was to prove to them that we were able to be more responsive than UST had been and that we were able to make their lives easier. Each time we dealt with a new developer, we sat down with them and explained what we were trying to accomplish and listened to their requirements. If they were apprehensive about any of the changes that we were planning to make, we proposed that they work with us to build the new environment, and that if they weren't pleased with the result, we wouldn't go production with it. (We have not yet encountered a case where the customer wound up taking us up on this.)

Build on Existing Strengths

WSG had several important strengths that we were able to build on for PSG.

WSG's greatest strength has always been its hiring and training practices. Instead of looking for candidates with extensive experience, they look for candidates who may have less experience but have good communications skills and the ability to learn. When a candidate without very much experience is hired, they work closely with existing employees to learn what they need to know.

This approach is important for several reasons. First, Unix administration can be taught, but communications skills and the ability to learn cannot be. Second, when existing employees work with new employees, it helps create a cooperative group culture. And finally, it alleviates the classic problem of hiring seasoned system administrators in an academic environment or in a tight job market.

Because PSG was initially a subset of WSG, we inherited WSG's hiring practices, and they became our strongest advantage. Much of what we have accomplished since would not have been possible if we had not been able to hire the right people.

Another important strength of WSG is that as a cost-recovery group, they depend on paying customers for continued funding, so they have developed a culture of good customer service. They also have the freedom to evaluate new work as it comes in and decline to take on new work if they feel that they can not do a good job in a particular environment. Taking this same approach in PSG allowed us to implement the group in an incremental fashion.

Incremental Approach

The most important part of this transition is that we've implemented it in an incremental fashion. For example, instead of PSG assuming responsibility for a large number of existing UST machines all at once, we waited until each machine was due to be upgraded. When that occurred, PSG worked with the existing service manager to build a new environment on a system with a standardized PSG configuration. Once the new environment was ready, it replaced the old environment in production, and PSG became responsible for it.

This approach has been important for several reasons:

- The existing UST systems had no standardization and no documentation, so PSG would have needed to reinstall them anyway. Waiting for an upgrade allowed us to avoid unnecessary service disruption.
- Because it took time for us to build our infrastructure, PSG had relatively limited capacity when the group was first formed. As we built our infrastructure, we had the capability to take on more systems. The incremental approach allowed us to take on systems as we felt we were ready for them.
- Each success that we had was visible to those customers that we were not yet working with. The enthusiasm of our existing customers made them very useful as references when we needed buy-in from later customers.

One Problem at a Time

One interesting point about the process of forming PSG and PAG is that we didn't originally intend to address the service management issue. We were only thinking about addressing our system management problems when we formed PSG. It wasn't until after those problems were addressed that the need for service managers became clear. In effect, our approach was to solve one problem at a time, since each solution helped us determine where to shift our attention next.

Forming the groups one at a time had several major advantages. First, we were able to focus all of our attention on each group as it was first being formed and developing its infrastructure. Second, we avoided a lot of organizational confusion which could have been caused by making two sweeping changes at once. Third, because PSG's need to define its relationship with its customers was one of the main factors that led to the formation of PAG, people were already familiar with the service manager's role when PAG was formed, so getting buy-in for the new group was much less of a problem than it had been for PSG. And finally, our success with PSG made it easier for management to obtain the necessary funding to create PAG.

Focus on Hiring Good People

Just as it was crucial to hire the right people to populate PSG, it has also been extremely important to have the right people in PAG. In particular, the manager of PAG was chosen from an internal search, since we felt that it was important for this role to be filled by someone who was already familiar with our organization.

In addition to the manager, PAG has hired three additional Service Managers from a mix of internal and external searches. They are individuals who have both technical expertise and project management skills. Because PAG is still in its infancy, it is too early to tell how heterogeneous the group will become. However, it is expected that the group will eventually be quite large due to the number of services that it will be responsible for, especially considering that multiple people within the group must be trained to back each other up on each service.

Pitfalls

There were several pitfalls in the process of implementing PSG and PAG. We anticipated and avoided some of them, but others needed to be handled on the fly. Thus far, none of these has been a show-stopper.

Root Access

The aspect of PSG's system configuration that caused the most concern from the developers was that we did not provide root access to service managers on our production systems. Instead, we worked with the customer to define the privileged tasks that they needed to perform on a routine basis and allowed those tasks to be accomplished via a mechanism like sudo [4]. Most of the developers were used to having root access on the production systems, and were reluctant to give up that access. These fears were usually based on previous experience with system managers who were not able to be responsive to customer requests. However, once we demonstrated to our customers that we did respond quickly to requests, they were all quite satisfied with the new arrangements.

The System Manager/Service Manager Interface

One of the hardest parts of defining the service manager's role has been explaining to people where the system manager's responsibility ends and the service manager's begins. In some cases, such as an LDAP server or a news server, the distinction is easy to identify; the LDAP or news server software and data are the responsibility of the service manager, and everything else is the responsibility of the system manager.

However, in other cases, the distinction is much less obvious. For example, in the case of a mail server, should the service manager be responsible for sendmail even though sendmail has traditionally been part of the system software? The answer to that question really depends on the details of the service, so we handle it on a case-by-case basis. Before taking on a new system, we will sit down with the service manager and come to an agreement about where the line will be drawn for that particular system and service.

There are no hard and fast rules for where we draw the line, but the most useful guideline is economy of scale. Whenever possible, a facility that is common to more than just one or two services should be handled by PSG to capitalize on our economy of scale. For example, PSG has a standard Apache installation that we can install upon customer request, since a large number of the services that run on our systems are web-based. On the other hand, we have one customer who does a lot of coding in Java, but we don't offer any support for this (other than simply installing the Java packages from the OS vendor) because none of our other customers require it.

The Service Manager/Developer Interface

The classic problem with separating development and production is that the developers don't have direct understanding of what makes an application easy to maintain in production. We hope to avoid this problem by carefully defining the interface between the Service Manager and the developers.

When a service manager determines that there are no acceptable existing software packages, he or she may contact a service developer to request that in-house development be done. The developer's customer is the service manager, who must give him a written list of requirements for the service. Based on those requirements, the developer is responsible for designing, implementing, and documenting the service software before handing it off to the service manager.

It should be noted that once the developer's completed software is handed off to the service manager, it should be treated no differently than if an off-the-shelf commercial product had been chosen. The documentation written by the developer should describe the installation and configuration of the software in a generic form, not the local settings and operational procedures actually used by the service manager to manage the service. The latter will still need to be written by the service manager prior to deployment.

Providing Job Satisfaction

It's been suggested that good technical people prefer a mix of fire-fighting and project work. While our environment is moving in the direction of avoiding the need for fire-fighting, we recognize that we can't anticipate every possible problem, so there will always be a small fire-fighting component of both our PSG and PAG positions. Also, because fire-fighting is only a small component of our environment, we have focused our hiring practices to find candidates who enjoy project work more than they do fire-fighting.

Another concern was that PSG's standardized system configuration would make the system manager's job less interesting. However, our ongoing work on new automation tools and standardized system configuration provides enough interesting job content to compensate for the lack of flexibility on individual systems, so this has not been a problem.

Thoughts About Other Environments

The process of implementing PSG and PAG was tailored for an academic environment like ours. In this section, I will theorize as to what might be different in an enterprise or ISP setting.

Getting Buy-In

In a commercial setting, management typically has more authority to issue directives to those working in the trenches. As a result, getting buy-in from management becomes a crucial part of the process. However, getting buy-in from the technical people is still crucial, since organizational changes will not work very well if people are forced to implement them against their will.

Implementing the Details

While a directive to create a service manager role may come down from management in a commercial setting, the details of how to implement the change should still be decided at a very low level. This would help people feel as though they have a hand in the decision-making process, which helps with getting buy-in.

Which Comes First?

In our environment, it was the formation of PSG that put the spotlight on the need for PAG. However, in an environment which already has a good group of system managers, it might make more sense to leave them as-is and focus immediately on creating a group of service managers. This should be fairly easy to sell to the existing system managers by focusing on the fact that the service managers will help ease their workload.

Change Control

One of the tasks that both PSG and PAG are working on is to implement change control procedures for our production systems and services. Our goal is to focus on the beneficial aspects of change control without becoming too mired in bureaucracy.

Many corporate environments already have complex, bureaucratic change control procedures in place. The creation of a group like PAG may allow the change control procedures to be simplified to maximize benefit and minimize overhead.

Conclusion

The organizational changes we have implemented to create a service management group have had excellent results. In the last couple of years, we have experienced a huge growth in the number of production systems we run. UST used to manage approximately 20 systems with five people; PSG is now managing approximately 85 systems with 4.5 people. The average turnover time for one of UST's high-profile positions was approximately one year; in contrast, PSG has existed for over two years and has never lost an employee. Emergency pages have gone from almost nightly to almost never. Finally and most importantly, the quality and reliability of our production services has improved noticeably.

Acknowledgements

I would like to thank Mona Heath not only for her proofreading, grammatical corrections, and suggestions on content, but also for her overall mentorship over the last several years. Without her support and encouragement, none of the events described in this paper would have been possible.

Author Information

Mark Roth is the manager of the Production Systems Group of the Computing and Communications Services Office at the University of Illinois at Urbana-Champaign, where he earned a Bachelor's degree in Computer Science. Mark is also the author of several open-source software packages. He can be contacted via email at roth@uiuc.edu, and his web page is http://www.uiuc.edu/ph/www/roth.

References

[1] Traugott, Steve and Joel Huddleston, "Bootstrapping an Infrastructure," *LISA XII Proceedings*, Boston, MA, pp. 181-196, 1998.

[2] Hunter, Tim and Scott Watanabe, "Guerrilla System Administration," *LISA VII Proceedings*, Monterey, CA, pp. 99-105, 1993.

[3] Limoncelli, Thomas, "Deconstructing User Requests and the Nine Step Model," *LISA XIII Proceedings*, Seattle, WA, pp. 35-44, 1999.

[4] Miller, Todd, "sudo," http://www.courtesan.com/sudo/ .