AT YOUR Service

Creating a service-oriented Software Engineering Team within your organization

by Robert Sabourin

Introducing new software engineering practices to any organization can be a challenge, even in the best of times. I don't want to break something that works, and I have to ensure that high-quality solutions are developed. I want to ensure that any process I have added or changed has improved productivity measurably.

QUICK LOOK

- **A light and effective software engineering model**
- **Launching a service-oriented approach in 8 simple steps**

If your job is to create, build, enhance, promote, reorganize, or *re-energize* a software engineering or quality assurance team, you need a light and effective process with a service-oriented philosophy. This will help you enhance productivity, encourage best practice, and achieve excellence without overburdening folks with overwhelming bureaucracy.

A light and effective process ensures that teams consistently develop software on time, on quality, and on budget—without stifling creativity or scaring away the good developers and product managers. A *light* process is one that can be implemented painlessly with a minimal additional effort by those using it. An *effective* process is one that has visible short-term results. A truly effective process will demonstrate value to the user quickly and naturally; its users don't have to have degrees in data analysis to understand that they're more productive thanks to using it.

If your job involves software quality assurance, testing, development, or product management, this article will give you food for thought. You may discover some ways you can simplify your daily workflow and be more productive at your job. We'll take a look at the basic practical steps required to

create, *from scratch*, a service-oriented Software Engineering Team (SET) or quality assurance group within your organization.

This method is applicable to many different organizations. Broadly speaking, I would classify these organizations as being (a) new to software development, (b) experienced in software development but new to software quality assurance, or (c) experienced in software development, with some testing but with no formal quality assurance or software engineering process in place.

Note that you may opt to implement these concepts outside of the confines of software engineering or quality assurance teams—depending on the priorities of the day and the existing team structures. Formal inspections, for example, could be implemented outside of software engineering.

Eight Steps Toward a Service Model

With several customers I have started to approach the introduction of software engineering using a simple *service* model, in which software engineering services are offered to existing projects to improve their productivity and help them consistently deliver quality products.

To succeed in business today, having satisfied customers is just not good enough—and the same holds true when you're an internal service provider. Your internal customers must be truly excited about your service offering; they must be "raving fans." (The term will be familiar to people who have read Ken Blanchard and Sheldon Bowles' customer service how-to book by the same name.) Enthusiastic customers are your best advertising, attesting to the value and quality provided by your product and services.

Here is a summary of the steps I recommend you follow in order to set up an effective SET in your own organization, in a way that will inspire raving fans of your own:

1. Define which services you want to offer
2. Discover which services your internal customers really need
3. Adjust service offering based on real needs
4. Establish core SET services
   – requirement flow
   – bug flow
   – formal inspections
5. Deliver above and beyond expectations
6. Add services incrementally
7. Get feedback from your internal customers
8. Involve people in all process improvements

To succeed in business today, having satisfied customers is just not good enough—and the same holds true when you're an internal service provider. Your internal customers must be truly excited about your service offering; they must be "raving fans."

At an absolute bare minimum, for my least technical customers, I insist on implementing a *requirement flow* and *bug flow* process. For those organizations, we leave out the steps dealing with formal inspections and more advanced services such as source control and configuration management. These can be added at a later date depending on the customer's technical savvy.

1 Define Which Services You Want to Offer

Some investigation is required at the onset to hone in on the SET service offerings most needed in a particular organization or for a particular project. Most organizations—especially those new to software development—don't know or understand most SET services, even though they most assuredly will need them. You must figure out clear ways to describe each of the services your team proposes to provide. Your SET services can include a broad range of offerings:

- requirement management
  – requirement flow
  – templates
  – examples
- test services
  – analysis
  – bug flow
  – testing
  – site monitoring
- formal inspections
  – moderation
  – statistics and documentation

As you implement processes, you will build up a library of forms, charts, reports, and checklists. You can use these to create standard templates, which will have several benefits: saving each project the effort of creating its own reports and checklists, promoting consistency, and creating even more raving fans. As you succeed in building raving fan customers you will find that project managers, developers, and other stakeholders who might have initially been reluctant to use your services will start asking for them.

Service Model

In setting up SET as a service to development teams, it is critical to ensure that internal and external customers are actually receiving something worth raving about: a service

that improves productivity at all stages. The three secrets of raving-fans customer service are:

- decide what you want
- discover what the customer wants
- deliver above and beyond expectations

You must first create a clear vision of what *you* want the SET department to be like at some time in the ideal future. How, when, and why will the internal customer be using the services offered by the team? Your vision should be specific, and related to every user of the services. I take the time to write down the vision in detail, and focus especially on how people feel the service benefits them in their work. The goal is that each time the SET service is used, the customers feel that they are more productive because of it.

2

Discover Which Services Your Internal Customers Really Need

First you need to know who your customers are. Your customers aren't just the leads and managers—they're anyone touched by your service. Possible customers include developers; testers; technical writers; project, product, test, documentation, and development managers; technical support and help desk staff; company executives; and other stakeholders.

What Do Your Customers Expect?

On an individual basis you should find out what your customers expect from your team (products, services, information, data, etc.). Be polite *and* be firm enough to get specific input. Avoid vague input by presenting alternatives to the customers. If the customers don't understand what you're proposing, help them by illustrating some of the benefits of what you'll provide. This is essential SET public relations; meet them in person, or at the very least speak voice-to-voice with them.

Different stakeholders may expect different things from the same service. Developers may want good, clear bug reports to help them find and fix associated defects, while help desk support staff want good descriptions of workarounds for any bugs we decided to leave in the product. Managers, on the other hand, may focus on knowing how many bugs must be fixed before we can ship.

3

Adjust Service Offering Based on Real Needs

Once you know what the customer wants, you should compare it with your own vision. Note that if there are wide gaps between your customers' wants and your ideal service offering—or if there are many gaping holes—then consider redirecting the customer to some other department or organization. *You should not try to be all things for all people.*

If expectations for any service are more than you can deliver, don't over-promise! Pick a reasonable subset. Implement it well. Add to it progressively over time.

A typical example of disconnect is a sales department expecting your SET to provide specific platform recommendations. If this is not in your service offering vision, then you should redirect Sales to Product Management (or whatever team handles platform recommendations). Making it clear that you don't provide such a service will enhance credibility for the services you *do* provide.

4

Establish Core SET Services

I will outline now the three basic service offerings that form the cornerstone of SET. The *requirement flow* helps clarify project goals. It is implemented first and answers the question "What are we doing?" The *bug flow* is implemented next, after you've established the requirement flow, and it answers the questions "Are we finished?" and "Can we ship yet?" With requirement flow and bug flow in place, we can define and complete a project, but we also want to offer services to help out with the stuff that happens between those two points: development. No matter how you develop software—in house, contracted outside, using third-party software, or a blend of these—the best SET service to establish next is the *formal inspection* program. Formal inspections can be used to identify defects in artifacts produced as part of the requirement flow, bug flow, and any other business or technical activities already taking place in your company.

Establish a Requirement Flow

What is the goal? Goals can be captured and stated as a series of requirements, which is the intent of *requirement flow*—the first process your SET should implement for its clients. Here every requirement is captured, clarified, written down, and prioritized. I try not to dictate who should decide on the requirement priorities, but I am very insistent that *on each project there is a well-defined way to prioritize requirements.* Requirements for a project should remain reasonably stable, and must be stated in a clear manner understood by all who use them—from nontechnical marketing staff to very technical development staff.

On each project one person is responsible for requirements. The job includes elicitation, collection, and identification of requirements from all the important project stakeholders, including customers and end users, as well as installation, deployment, development, and testing roles.

For each requirement we ensure that the following information is captured:

- unique identifier
- functional area
- source
- description
- additional information and references
- implementation priority

The priority for implementing a requirement is set to one of four values, depending on when the requirement should be implemented:

- current release
- next release
- some future release
- never

Every time a new requirement is added, its priority must be established relative to the other requirements based on the current business context. If the new requirement changes the project scope but generates considerable new revenue, then it may have a high priority relative to other requirements.

Reassess the priority of all requirements, independent of their currently assigned priority, whenever the business context changes.

With the requirement flow process in place, we can react to evolving needs in a pragmatic (rather than chaotic) way. Requirements can be referenced all through development, and we improve productivity by removing ambiguity as to what project goals are. We can answer the fundamental question in software engineering: "What exactly are we trying to do?"

Establish a Bug Flow

How do you know you're finished? A critical service of the SET is testing—a basic service to ensure that development efforts conform to requirements.

The SET offers test analysis based on its understanding of the project requirements and the business and technical environment. Business (commercial) and technical (development) risks must be considered, and the highest-risk areas tested first. Testing those areas might include methods such as detailed analysis, while lower-risk areas may be tested with exploratory methods such as those based on the work of James Bach (see the StickyNotes feature at the end of this article).

Establishing bug priority and severity is key to knowing what's "good enough." (Or as the Bard might say, "To fix or not to fix, that is the question.") Setting the priority and severity of a bug is a business decision, and changing business conditions clearly have an impact on the priority and severity of a bug. The prioritization of bugs can only be done given full knowledge of the relevant business drivers associated with the product. In order to converge a project effectively, the process of bug reviews must be well defined and followed rigorously.

> Don't try to jump from the current state to the ideal vision in one step; in this plan, baby steps are the order of the day.

As few people as possible should be involved in the decision. In fact, three is ideal: test lead, development lead, and customer advocate. Effective business decisions will ensure that the product is released based on consistently applied rationale; it's very important that the same decision-making process be applied to all bugs of the system.

And as with requirement prioritization, if the business conditions of a project change, it's very important to review any passed decisions about bug prioritization—and re-prioritize if necessary.

With the bug flow process in place, we can understand the state of the software being developed, and our measures can tell us how many bugs must be corrected before we can ship the software. With this process in place, we now have a way to decide which bugs to fix and which bugs to keep—and know when we're truly finished.

Set Up a Formal Inspection Process

Formal inspections offer the biggest "bang for the buck" in terms of improving productivity, minimizing cost, and identifying defects early in the development cycle.

Your team can offer a formal inspection program that will allow its client projects to identify defects early in the development process. Just about any document can be inspected. The methods of Tom Gilb and Dorothy Graham, as described in their book *Software Inspection*, have been successfully used to identify defects in contracts, requirement documents, test plans, development plans, source code, and almost any other artifact of the development process imaginable. The SET staff can offer the administrative support of managing and collecting all necessary metrics for these formal inspections.

Generally, documents can be inspected as part of their creation process when the author feels the document is ready and complete, and the inspection process can be implemented in parallel to any development process in use. The SET's role can include moderating inspections and formally certifying moderators.

The major benefit of formal inspections is to identify defects early—well before development starts when used on specifications, requirements, plans, and design documents. Besides identifying important defects, the results of formal inspections may also be used to generate checklists of best practices and problems to avoid.

5

Deliver Above and Beyond Expectations

With a vision in hand, establish a strategy that will allow you to deliver consistently and extraordinarily—a standard of service Blanchard and Bowles' book describes as "deliver plus one percent." You have a vision of some ideal day in the future when you will have achieved the service

levels you want, but to get there you must offer services in small manageable chunks. Don't try to jump from the current state to the ideal vision in one step; in this plan, baby steps are the order of the day.

On each project, implement some process change that brings you closer to the ideal. In your bug tracking system, for example, start with bug descriptions targeted to development staff. When programmers can quickly find and fix defects based on your descriptions, then you're ready to start adding new information such as workaround tips for end users.

Innovate progressively. And don't forget to frequently update your vision and reconcile it with the needs of your internal customers.

Inconsistency Can Destroy a Lot of Built-up Goodwill and Productivity

You must be able to offer your services with consistency in order to build credibility and generate raving fans. If your customers experience too many inconsistencies, they won't know what to expect—and that will lead to frustration. Your SET has to provide a stable level of service in all projects, independent of who on the team was responsible for the deliverable.

An excellent example is found in bug report descriptions. You may have one junior tester taking great initiative and helping out in an area unfamiliar to him; but if that means that your team's reports vary depending on who wrote them, the whole team may appear incompetent. While it's always a delight to encourage team members in their initiatives, find a way to do it that doesn't impact the team's consistent fulfillment of its deliverables.

First Meet, Then Exceed, Customer Expectations

Don't try to add all sorts of new process steps or deliverables until you can consistently deliver what is *presently* required. For example, the first step is to offer a bug graph updated once a week. The bug graph must be consistently accurate, punctual, clear, and available to all users. Once weekly bug graph updates have been perfected, then you can move to daily updates of the bug graph. Only after perfecting the daily bug graph should you consider offering on-demand, real-time bug graphs.

If the customer expects a great test plan outline with coverage of all features, ensure that this expectation is consistently met before adding requirement tracing or usage scenarios. Requirement tracing, for example, can be a very powerful tool to add to project test planning and test case design—but you shouldn't add it to your SET system until you can consistently manage requirements and test plans.

6

Add Services Incrementally

Add services to your portfolio gradually, based on evolving priorities. Ensure that productivity is improved progressively. If you ever miss the mark (by accidentally implementing a service that diminishes productivity), remove the service that's gumming up the works.

Keep your Sherlock Holmes hat on at all times. Investigate and use a simple combination of observation and elementary deduction to identify potential new services. There are two sets of useful questions to ask when looking for additional services to provide.

First, could the same service be used by a different internal customer?

- Could a testing service be used by product managers to do competitive analysis?
- Could formal inspections be used by the legal department to help develop better contracts?
- Could bug tracking be of use to the customer service help desk?

Second, could we offer new (related) services to our current internal customers?

- Could configuration management be used to manage project documentation?
- What about reviews?
- Code walkthroughs?
- Facilitating requirement gathering meetings?
- Facilitating project post mortem review sessions?
- What information metrics can we provide to project stakeholders?
- Can we extract software complexity metrics from our configuration management system?
- Can we provide work effort breakdowns by project and activity?
- Can we package information in a new way to make it useful to another potential internal customer?

Keep in mind that you should be consistent in your current offerings before adding too many new services. I suggest challenging your team to add, change, or remove at most one new SET service per project.

7

Get Feedback from Your Internal Customers

Get feedback from your internal customers about how they enjoy and appreciate the service offered. Are you generating raving fan internal customers?

SQA teams are expert in measuring all sorts of aspects of the software product under development. Measures should *also* be put in place to determine the effectiveness of their work. Measures should be designed to show how customers feel about the service. Was the service delivered the service that was required? Was it offered in a timely manner? Was the service complete? What if anything was missing? What did the customers like best? What did they like least? How can we serve them better next time?

Do they come back for help, advice, and guidance? Do they *use* the deliverables your team is creating? Are there some deliverables not being used? If not, why not? Should we not offer that deliverable in the future? Are our customers excited? Are they having fun?

There is no magic trick here—you just have to go out and get this feed-

back! Ensure that soliciting feedback is part of the SET service offering, and burn that into the mentality of the team. Have everyone solicit feedback on completion of (and *during*, as appropriate) implementation of a task. I have seen examples in which the wrong service is offered due to misunderstandings between the SET and development team. Actively solicited feedback can catch these problems early.

Evaluate your SET's score at the end of a project. How did SET do in generating raving fans? Did SET consistently exceed expectations? What areas need improvement? What areas are excellent and should be encouraged? How can we apply this excellence to other service offerings?

8 Involve People in All Process Improvements

In the end, it's more about people than about paperwork; be sensitive to their needs as you plan your approach. It seems to me that there are two ways to implement process improvement. You can parachute in new process and scrap everything in your path, a sort of "big bang" approach, or you can implement process improvement in baby steps.

Decide what you want to offer, discover what the customer wants, and deliver plus one percent—that's the key to implementing process in an environment without an established software engineering process.

I have had the most success with the baby steps approach—involving people in the process evolution, and making sure goals, roles, and responsibilities are clear and that they evolve. Baby steps work especially well on short development cycle projects, such as those dominant in today's Web and e-commerce application development. But they apply equally well to any major development project implemented in a phased approach; each phase of a large project could be used to introduce process improvements in manageable chunks. Just make sure the team knows the goals and is aware of how each baby-step process improvement will improve productivity.

Getting Things Done

Companies cannot drop everything to implement software engineering process, and are very rarely motivated to do so. By introducing software engineering process as part of a *service-oriented* model, we allow each and every project to improve progressively as staff buys into the process improvements being implemented. The goal is to support the development teams by providing a service they can use to achieve consistent on-time, on-quality, and on-budget projects.

Using the steps we've outlined here, light and effective process can improve productivity without much pain or hassle. Decide what you want to offer, discover what the customer wants, and deliver plus one percent—that's the key to implementing process in an environment without an established software engineering process.

In organizations in which Management might be tempted to see the software engineering team as either a necessary evil or a project bottleneck, establishing this kind of service-oriented SET is an effective way to improve productivity and quality without disrupting any ongoing development. When you can turn skeptics into grateful and satisfied internal customers, your team will be recognized as improving productivity for everyone. STQE

Two Caveats

The use of SET services cannot be forced upon development teams. If you shove the service down people's throats, then you are effectively acting like a big bad corporate monopoly. Folks will not buy into the offering, and you will not generate raving fan internal customers. The development managers should be encouraged to work with SET in defining a reasonable and useful service offering.

Factor the status quo into your SET plans. If your company already has some well-established, traditional, software engineering or quality assurance process in place, then you must be very careful in introducing a service-oriented SET model. Make sure that any new SET services won't conflict with established practice. If you want to evolve from the status quo to a new SET model then it must be done in baby steps, avoiding productivity loss and culture shock. In this case, I would recommend introducing the SET concept for a service offering that does not yet exist in your organization (formal inspections might be a good example). Get folks comfortable with the service idea before changing the way they implement process-related tasks that are established and well understood.

Robert Sabourin (rsabourin@amibug.com) *is an author, lecturer, and president of AmiBug.com, a firm specializing in software management consulting, teaching, and professional development. Robert is the author of the popular children's book* I am a Bug *(ISBN 0-9685774-0-7), which explains what SQA folks really do at work.*