

## The Hidden Costs of Application Development ... Using Process as a Productivity Tool

By Mac Felsing

My intent, over the next few months, is to offer perspectives on process in this newsletter. As with anything in business or life, our needs, wants, and other requirements are most often based upon our perceptions.

### Executive Perspective

IT Executives are focused on the business of running a company. They are interested in what impacts stockholders equity and interest. They look for ways to increase profit margins, achieve competitive advantage, improve quality, lower operational costs and reduce time to market.

### Management Perspective

Managers are looking for ways to deliver the products and services that are under their direct control. Management objectives include cost of goods and services, delivery, customer satisfaction, employee productivity and satisfaction, as well as security. Managers and employees are often evaluated or rewarded based on their ability to produce a specific deliverable in a specific timeframe -- often times, with inadequate budget, causing compromises in quality. "What do you mean? Zero Defects"

### Developers Perspective

Developers on the other hand, are just trying to survive. They spend as much time as they can doing what they love – creating code, applications, solving neat problems with elegant solutions, avoiding their managers and executives. Developers are skeptical of new processes that "will fix all of our problems" when in actuality; they waste most of their time, working on tasks that are not central to their main function, which is creating enterprise class applications. Next they are asked to document their work, which is never a high priority, and always done at the end of a milestone for the project. This effort is not fun and usually not adequate for the team that will maintain the application.

### Process Perspectives

In this series, we will provide Executives, Managers and Developers with some insights on what to look for when setting up or changing your software development processes as well as some of the early warning signs that there may be a problem.

Based on our experience, we would like to address a few key areas that can help or hinder an organization's success when it comes to process. Each point will be supported by various examples along with commentary on how they would be addressed within Feature Driven Development.

### Definitions

To begin, let's define two terms so that we are all on the same page for the rest of the series.

**Process** - A process is simply a set of tasks or activities that accomplishes a goal (i.e. does work). In the world of application development effective communication is a critical part of the process. Developers work on certain aspects of a project but how does management know what stage of development the application is in? Executives hold

ultimate responsibility for the finished project. How do they know what to tell their customer? Process provides this information.

Philip Crosby stated, "All work is a process".

A process may be formal or informal, heavy or agile, general or specific, ad-hoc or well known.

**Noise-level** – In the context of processes, I use this term to refer to activities that are so pervasive that they are assumed to be an integral part of a process. These activities have a very high cost both in time and money. Reducing the Noise-Level quickly translates to ROI (return on investment) while improving communication and employee satisfaction. While looking for Noise from development teams listen for statements like "we've always done it this way". Just like ambient sounds in an office or on a street or in a crowded restaurant. I also refer to individual activities in the noise-level as **time stealers**.

Whatever you refer to these activities as, they are the ones that cost you and your team money, time and your reputation for delivering high quality applications on time.

## Impact of Activity Noise-Level

Just like static noise on a radio can interfere with the listener's ability to hear the music, if the activity noise-level in an organization is too high, it can prevent or impede your team from achieving its project goals.

The noise-level represents an amount of activity (work) that is being performed within your organization that is usually not tracked.

We've all been through planning exercises for scheduling projects over a given time period, (annual budgeting exercises are typical example), calculating the number of project hours and the number of resource hours we have available or can add. There is usually some type of available man-hour formula and an estimate of all the candidate projects. The man-hour formula could look like this:

$$\text{numberOfStaff} * (\text{workHoursPerWeek} * \text{numberOfWeeks}) = \text{amountOfAvailableHours}$$

- The formula can be modified to account for holidays and vacations.
- The workHoursPerWeek can be adjusted for actual work time. This presumes that an 8-hour workday actually contains around 5.6 productive hours.
- Then the number of projects you want to work on are estimated.
- The projects are prioritized, and put on the list of approved projects until the amountOfAvailableHours = 0.
- Arrangements may be made to add additional personnel to increase the amountOfAvailableHours thereby increasing the organizations capacity

### *So what's the problem?*

The problem is the noise-level. Here are a couple of examples. You can probably think of many more.

## Sample Activity Noise Level

**\*Note:** *"There is no such thing as a 5 minute change!"*

## # 1 – Adding Functionality to an Existing System

A developer is assigned a task to add some functionality to a system. Let's say the application is a C++ or Java application, and the functionality is in 6 or 7 files. Let's also say that the process does not include a strict set of controls for version control. Let's say other developers are modifying half of the files. This task is a priority and must be done quickly because it's for your biggest customer. Sound familiar?

### *Where is the noise?*

- 1) The developer has to find the latest version of the 7 files that will be modified. This takes time; say on the average 10 minutes per file. Why? The reasons are many; the last person to modify one of the files did not put it back in the right place (version control or in some "golden" directory), or a copy was taken out and someone is working on it, but didn't check it out for modification, or the developer has to track down who has the last copy. Up to 70 minutes will be wasted by the developer to get the correct files to be modified.
- 2) The developer makes the necessary modifications and compiles the application (not realizing that a couple of the utility libraries being used are out of date), runs some tests and submits the changes. You try to deploy the application to test, but it doesn't compile (remember the obsolete utility libraries?). It is sent back to the developer. Reasonable estimates for elapsed time is 3 hours.
- 3) The developer figures out the problem (30 minutes), changes the libraries, recompiles and tests, makes a one-line modification (5 minutes\*), re-tests (15 minutes), and submits the code back in (10 minutes). System test went home for the day, so it waits until the next day before being recompiled (1 and ½ hour delay – not counting the lost day), etc.

There are a lot more steps to this story, but it is enough to show the effect of noise. You may be saying, "When did you visit our development shop?" or "This happens all the time. What's the big deal? Let's take a look.

Step 1 took 70 minutes.

Step 2 took 3 hours or 180 minutes.

Step 3 took a minimum of 60 minutes and delayed work by another 90 minutes.

---

Total time "stolen" = 400 minutes or 6.6 hours.

And we didn't even talk about other developers loading their changes over the changes our example developer made!

***That's the normal cost of business, right? That's the way development works, isn't it. It was only one developer, right? Usually. No. And no.***

Let's consider that the above scenario takes place once a week per developer and you have 30 developers. Now we are talking about 198 hours, or about 5 man weeks of time. And that's for one week! That's time that you can't get back on the project! Could you use that time to deliver more features? Could you use that time on additional projects?

With some attention to the procedures and maybe some tools, Step 1 could be reduced or eliminated, step 2 could be automated and reduced to a few minutes and step 3 would be eliminated! ***This would cut the 6.6 hours down to 10 or 15 minutes.***

## #2 – Project Status

How many times have you had to stop what you were doing because the project sponsor or another executive needed to know the status of a project and he needed it in 15 minutes for a very important meeting.

You dropped whatever you were doing, went to your direct reports, or worse, broke the reporting chain and went directly to each developer to find out the status of their work. Let's say you had 5 direct reports, they each had 5 direct reports or a total of 30 people. Let's imagine that in order to get the status, you take 10 minutes for each person, to get up, walk down the hall, track them down, corner them about the project status, and get them to drop what they are doing to get it for you. Then your direct reports have to do the same with each of their developers, testers, etc.

	Managers	Direct Reports	Time	Total Time
Request Status from Direct Reports	1	5*	10 min	50 min.
Direct Reports get status from each developer	5	5*	10 min	250 min.
Assemble status report from data				30 min.
Total time to collect status				330 min. or 5 ½ hours

A total of 5 ½ hours was needed to collect status. How many times in a week or two week period have you been asked for status? How much time is your team spending recording status? (That is another variation of this non-productive time-stealing task).

With a little planning and organizing, you may be able to change the process. Imagine if the status is automatically generated as a function of completing a task or milestone, thereby reducing the amount of time it takes to gather and report status. This is a key technique to reducing the activity noise-level in an organization. It then becomes a mechanical exercise gathering the atomic task information and rolling it up to project management and executive levels. This also increases the accuracy of the reporting because it is a more accurate representation on the work that is completed.

***Wouldn't it be nice to have a place where people could go to find out the status of a given project without interrupting your schedule?***

### Managing activity noise-level

The activity noise-level can have a very real impact on the organizations ability to meet its business objectives and goals. Learning to recognize, identify, and correct noise-level activities can have a tremendous positive impact on the organization's ability to deliver projects on time, within budget, with the asked-for features.

Here are a few indicators that the activity noise-level may be adversely affecting your team.

- 1) Simple tasks take significantly longer to do than expected or don't get finished.

- 2) Tasks aren't completed because you are constantly putting out fires, chasing down information, and doing a lot of busy, unproductive work.
- 3) The information needed for completing a task resides in several areas.
- 4) The next step in a process gets delayed because notification that the task was completed is missed, not sent, not expected. Or not everyone on the notification list was notified.
- 5) Tasks completed by one group are moved to the next group and returned because something simple was missing.

If you have any of those symptoms in your organizations, start zeroing in on the specific activities where you notice the symptoms. Here are some ideas to get you started.

- 1) Observe the activity in question. Does the activity involve getting information from multiple places, involve phone calls to different people, or require a response or permission from someone? How many times does this occur?
  - a) As a possible remedy, consider collecting the information in one area that is accessible to all the people that require the information, or a least a single access point. Determine the period of time needed between refreshes of the data in that area, (e.g. weekly, daily, hourly, continuous).
  - b) As an additional exercise, before any changes are made, measure the amount of time the task requires and how many times it is done in a finite period of time (1 day, or 1 week). This will take a little effort. Then make the changes, and repeat the measurement.
- 2) Observe how many times work must be redone, usually in areas where transitions occur, such as moving from design to code, or from coding to test. What is the rejection rate (the transition fails), because the receiving group was not prepared to receive the work?
  - a) A centrally accessible area can be established to track the state of each artifact (requirements, design documents, source code, test cases, etc.) as it moves through the various phases of the software development life cycle (SDLC). This might take the form of a web page or pages with links to the artifacts, or could be as simple as a bulletin board, if the team is physically located in the same area.
  - b) Using a system (workflow engine software) that generates automated messages that notify team members when an artifact is ready for the next step, or even notify the next person responsible that there is a task to perform on an artifact can significantly reduce the number of times the artifact is handled and the dead-time between activities, which will reduce the overall duration and cost of development.
- 3) As an artifact passes through various steps in its life cycle and as tasks are performed and completed using the artifacts, one method for insuring that each critical step or transition in the process goes smoothly, is to set up exit criteria for at least each milestone. You can then use the exit criteria to set up a checklist of the artifacts that have to be completed for that milestone. Each role responsible for ensuring that a milestone is delivered is also responsible for completing the checklist for that milestone. This can be formal or informal, depending on your specific organizational and project requirements.

## Corrective Action and FDD

As demonstrated above with the general examples, corrective action can be taken without a specific process. None of the possible remedies mentioned above are new. They have all been studied, used, and proven over the years. However, if you take a little more time developing a solution, you can come up with several best practices that work together in a synergistic fashion that end up accomplishing a whole lot more than any one of them. This is where some of the Agile Processes come in, particularly, one known as Feature Driven Development or FDD. FDD features a number of best practices that work together in just this manner. One added benefit of using these best practices together is that it naturally reduces the amount of activity noise level in the software development organization.

With a little planning and organizing, you may be able to change your organization's existing process or augment it with steps included in another process. One example of this would be the use of aspects of the agile process Feature Driven Development (FDD). Feature Driven Development (FDD) focuses on small cycles, small deliverables with user identifiable functionality and ongoing feedback. Remember, if you only implement part of the FDD process, you will only see part of the benefit, however, this can be one way of implementing the process in your organization.

One way FDD helps to reduce some of the activity noise level is by using milestones to measure progress. Milestones are nothing new, but the discipline applied to using milestones in FDD makes a big difference. During the construction phase of FDD, which is Design by Feature and Build by Feature, there are a total of six milestones that mark progress on each feature. The milestones are binary in nature. They are either complete or not started. Remember, there are six milestones on each feature, which is time-boxed to two weeks or less. This means that the milestones are measuring small amounts of work and there is no guesswork. All six milestones must be completed before the feature is done. This makes the status reporting very accurate and also provides a basis for predicting future completions based on current performance.

Example: If you have 40 features to deliver in four weeks, that breaks down to completing 10 features a week, or 20 features every two weeks. If I just delivered 20 features but it took four weeks to do it, what is the probability that I will be able to complete the remaining 40 features in the allocated time if nothing else changes?

In our status-reporting example, using features and milestones, it becomes a relatively painless mechanical exercise to gather the milestone information and roll it up to project management and executive levels. This reduces the amount of time it takes to get project status and also increases the accuracy of the status report because the data is a more accurate representation of the completed work. There aren't any 90% completed programs or tasks that stay that way for six months!

What we have just demonstrated using a concept in FDD is a *key technique* to reducing the activity noise-level in an organization; **Automate mundane, time consuming tasks or make them a result of a value producing task, or both.**

Another aspect of FDD that works with the milestones is that each major step of the process uses a technique known as ETVX or Entry, Task, Validation, and eXit. This is a technique of defining the entry criteria for each step, describing the task or tasks

involved, identifying how the tasks can be validated, and listing the artifacts that must be produced as a result of the activities. This is done for each of the five major steps or phases within FDD<sup>1</sup>. This technique can be extended to each of the milestones as well.

For example, let's look at milestone number five in FDD, **Code Inspections**. How do I know when this milestone is complete? What items need to be inspected? What measurements or standards do I use to ensure that what I'm inspecting is complete and meets the level of quality required? In FDD, the Chief Programmer, sometimes referred to as the Feature Architect or Feature Owner, leads the code inspection. The Chief programmer is responsible for ensuring the code inspection is held. He or she needs to ensure that the delivered code will build and compile, that all of the delivered unit test cases have been run, that the test cases are valid and complete, that the code meets the organization's or project's coding standards, that the code meets the design criteria, that any notes or other packages needed to properly build and compile the feature have been included, and that the delivered code performs the expected functionality.

If that sounds like a lot of things to verify, you would be right. There are probably a few more things that could be added, based on your organization's needs. The sensible thing for the Chief Programmer to do is to make a little checklist. The checklist is a listing of the exit criteria for the milestone. It can be electronic (spreadsheets with hyperlinks to artifacts work very nicely), or on a piece of paper. It's also a nice thing for the Class/Code Owners working on the feature to have, to ensure that they deliver everything before the code inspection is scheduled, and to identify anything that needs to be corrected. Once everything has been approved in the inspection, all of the artifacts can be moved to the next step. (Hint: include the checklist as the entry criteria for the next milestone.)

If you have several Chief Programmers or team leaders and several projects that you are working on, let the CPs get together and create a standard checklist for each step they are responsible for. This will get buy-in from the CP's, and create a standard (base starting point) for all projects and groups.

#### Example Code Inspection Checklist

<b>Feature Name:</b> Make Reservation for Passenger		<b>CP:</b> JMF
		<b>Inspection Date:</b>
<b>Date completed</b>	<b>Item Description</b>	
	Code	
	Audit Passed	
	Meets Project Standards for MyCorp	
	Matches Design	
	Checked in to VCS	
	Build Notes Included	
	Build/Compile Successful	
	Unit Test Cases	
	Test Cases Validated	
	Test Cases Passed	
	Documentation	
	Class Diagrams	
	Sequence Diagrams	
	Use Cases Updated	

**Note:** The above is just a simplified example. Other information may be included, (i.e. class names, code location, class owners, etc.), as determined by the Development Manager, Project Manager, Chief Programmers or other appropriate decision makers.

There are many aspects of FDD that contribute to a lower overall activity noise-level in an organization. These were just a couple of minor examples. The advantage with FDD and other agile processes in this situation, is that having some type of predictable but flexible method for dealing with business problems, where communications and expectations are managed in some way, goes a long way towards reducing or eliminating the unproductive, noisy, time-wasting activities that occur in almost every software development environment.

By taking an active role and promoting an environment of continually improving the way work is being done in your organization, you encourage your greatest resources, the people that do the work, to be on the lookout for ways to improve your business and development processes. All of this contributes to a strong return on investment (time and energy well spent), while improving the way customers perceive your team's ability to deliver (affects the bottom line – profitability).

## Summary

We have examined some specific examples of time wasting activities found within the everyday tasks. We have looked at some alternatives that can reduce or eliminate the amount of time spent on those activities, and finally, have examined how an agile process like FDD is designed to minimize those types of activities by employing well known best practices in unique combinations to maximize productivity and allow teams to focus on producing results.

We also looked at a key concept in increasing productivity in an organization or process;

***Automate mundane, time consuming tasks or make them a result of a value producing task, or both.***

Paying attention to the processes your organization uses, and removing the obstacles and time-wasting tasks that stand in the way of producing results is an excellent way to increase the organization's productive capacity with a minimum of expense. Agile processes, in particular, Feature-Driven Development, offer some very specific techniques for reducing the activity noise levels, while providing predictable, repeatable methods for producing quality results.

<sup>1</sup>For more details, tips and hints about implementing FDD, see "A Practical Guide to Feature-Driven Development", Palmer and Felsing, Prentice Hall, Feb 2002