# Doing More for Less in Software and Solutions Development

By Neil Potter and Mary Sakry

When the economy gets bumpy, it is natural to tighten budgets, make cuts and expect the organization to work harder so that more can be produced with less.

Memos and emails are sometimes used to announce to an organization that reductions in cost are needed and that any extra effort by staff will be appreciated. The impulse to send a memo is understandable and can help, but might only have a temporary impact on the organization while the "save money" request is still fresh in people's minds.

For example, let's suppose the initial "save money" memo causes the group to delay buying hardware and software upgrades (e.g., $30K), stop printing emails on paper (e.g., $30/month) and reduce travel expenses (e.g., $10K/month). These are good ideas, but there is more that can be done to save money long term.

A systematic series of improvements is needed to produce more for less consistently over time. These improvements need to address inefficiencies that happen every day in projects in addition to cutting hard expenses.

Before we look at a summary list of improvements, consider the cost of a typical inefficiency that you might have. For example:

- Potentially avoidable coding mistakes enter system test causing the execution of an extra test cycle. The testing cycle might involve 5 people, testing for 5 days (6 hrs per day) at a company cost of $100/hr[1]. Potential inefficiency caused by one additional cycle = $15K.
- One additional month of testing is needed because there are no clear requirements to base testing upon. If testing could have been reduced by one month due to less guesswork and retesting, the potential savings could be: 1 month saved x 10 testers x $100/hr = $120K (working 20 days per month at 6 hours per day).

These are just example numbers. There are a lot of other numbers one could play with that show various levels of permanent savings. At $100/hr, almost any reduction in the effort expended, while maintaining output and quality, will add up quickly.

Since 1989 we have seen many companies that have been improving to create more with less. Below we share a summary of some of those improvements. Use the list to identify a few improvements that could benefit you.

**Produce more**

*Producing more* means producing *more usable functionality.* Some improvements to consider are:

- Defining user requirements using Use Cases, thereby building functionality that meets user needs and reducing the likelihood that unused functionality will be created.
- Implementing strict requirements management and not allowing a high volume of changes to be accepted without evaluation, replanning and negotiation. Allowing continual changes to enter the project with no evaluation can cause a level of chaos and prevent *any* feature from being completed.
- Writing a requirements document that mimics the user guide, thereby enabling more teams within the development cycle to use it. For example, a Use Case document can be read and approved by customers, used by developers and testers, and can form an early version of a user guide.

[1] $100/hr is an average company cost to employ one engineer. It includes benefits, heating, lighting, equipment and wages. Ranges vary widely across industry.

- Performing validation on selected requirements using customer reviews, models and prototypes. Not all requirements are necessarily useful or understandable when they are first written. Validation methods used at the requirements phase can provide inexpensive early feedback.
- Identifying and building reusable components that can be used across multiple products.

Producing more output can also be achieved by spending less time on rework and firefighting.

**Spend Less**

*Spending less* means spending less effort and cost to produce the same result and quality. Improvements to consider are:

- Managing risks. Risks are potential problems, and if they become problems can be very expensive in time and money to resolve. Identifying what could go wrong, setting priorities and taking actions to make risks less likely and less painful allows the project to keep moving forward. Risks should be assessed and tracked every one or two weeks.
- Spending less time transitioning between multiple tasks. If any resource has more than three significant tasks at one time, assume they will spend time putting one task down and picking up the next. This amount of time will vary, but could be up to 10% transitioning between each two tasks. Reducing the number of parallel tasks or sequencing some of the tasks reduces the transition tax.
- Identifying and removing non-value-added tasks and documentation. Examples of potential non-value-added tasks are: reinventing instead of reusing, using overly complicated process templates, voluminous project status reporting (e.g., reporting status two times per day), daily builds when nothing has changed, redundant test cases that test the same requirement, detailed design descriptions for trivial functions, and copying requirements text from document to document instead of having a central reference.
- Spending less time on rework and allowing time for more development of new functionality. Verification steps, such as requirements analysis, peer reviews and component testing are inexpensive techniques to find defects earlier and reduce rework. Any reduction in rework will allow more to be created for less.
- Reducing the amount of defects and problems that enter an unavoidable bottleneck. An unavoidable bottleneck is where a handoff occurs between two groups, and the capacity of the second group is less than the first. Examples include the handoff of components to test, integration, validation, final build, and approvals during the project. One organization reduced wasted integration time by using scorecards to evaluate and reject incomplete components entering the integration phase.
- Spending less time looking for documents instead of using a defined project folder system so information can be readily found.
- Using project phase reviews at key milestones to determine if a project is ready to move ahead. Simple criteria are established to assess whether a milestone is actually complete (e.g., for the coding phase: code has been inspected, reworked and placed under Configuration Management).
- Reducing email and meeting time by:
  - Planning communication ahead as scheduled tasks, specifying who needs to know the information, what information is needed and when they need it. When team members know that a communication event is coming up, they don't need to send lots of email in between time.
  - Conducting meetings with an agenda, moderator, clock and a projector to record minutes real-time.

Spending less time can be achieved by refining skills to work faster.

**Work Faster**

*Faster* means producing the same result in either less effort or calendar time. Improvements to consider are:

- Using, practicing and refining *best in class* skills to complete work faster. For example, eliciting requirements can take months if the wrong questions are asked of the wrong people. It can take days or weeks if a toolbox of requirements practices is used to select and interview end users.
- Using team members from different disciplines working together to elicit requirements, thereby reducing the amount of time spent playing *Telephone** to communicate requirements issues. For example, having a collaborative team consisting of end users, the project manager, a requirements analyst, developer and tester working together to collect the requirements saves clarification time downstream.
- Identifying tasks in the schedule that have no dependencies and moving them into timeslots when the team cannot work on anything else.
- Using time zones (if your team is in more than one geography) by defining work to be done with clear definitions (or interfaces), and planning communication events to integrate work together and resolve issues. When work is poorly defined or communication is haphazard, multiple time zones act as a sinkhole for project effort.
- Planning for incremental releases of functionality to enable early testing to occur on high-risk or core components.
- Conducting status updates with senior management, highlighting issues and risks that need quick resolution by management. Leaving management in the dark either causes very large surprises late in the project, or allows a project to realize a risk that could have been avoided.
- Using periodic reviews involving the key stakeholders to synchronize work across multiple dependent projects. In essence, this is designing the communication events into a project upfront, thereby reducing surprises, delays and downstream rework. For example, in some large projects, the software, hardware, test and senior managers meet every two weeks to review the schedule and risk management plan.

The improvements listed are only a small example of what can be adopted to produce more for less. Use these examples as a springboard to find opportunities in your organization.

For more information regarding our services, *please email help@processgroup.com or visit www.processgroup.com*.

*****Telephone: The players sit in a circle, and the first player whispers a phrase into the ear of the next player. This second player whispers it to the third player, and so on. When it returns to the first player, that player announces what the phrase has become and how it started: the two versions are often very different.