

# PARASOFT WHITE PAPER

Topic: Productivity vs. Quality: Achieving measurable results through proper workflow management

*Very often productivity and quality are distinctly at odds with one another.*

**When you develop in C or C++ –in particular, an embedded application–Parasoft C++test provides significant assistance. It offers a variety of techniques that allow you to effectively eradicate bugs from your software. However, when you start working in a team, you face challenge even beyond bugs creeping into your work. These days, when “simple” microcontrollers start to have megabytes of memory and have sufficient power to run multimedia applications, building the expected level of functionality into these devices requires a team of professionals—not just an individual. For example, software running on a moderately-advanced copy machine can have as many as 5 million lines of code.**

When your team is just a few people, then most likely all of your team members know who is working on what at any given time. However, you may not know how the work is being done (or, in other words, how the code is being written). As a result, when you finally build your application, it may experience numerous problems and you’ll be spending a lot of time trying to make it work well. Your productivity as a team sinks, and job satisfaction follows. To top it off, your company’s costs increase as well.

To prevent this, teams try to adopt best practices like coding standards or code reviews—or even pair programming or test driven development—relatively early in the development process. However, in the real world, those initiatives tend to deteriorate.

You find yourself doing such practices “occasionally” and resorting to excuses such as “no time,” “bad tools or lack of them,” or “lack of understanding from the management.” This is all true... isn’t it?

The fact is that there is now pressure to deliver more and more features in a shorter and shorter time frame, and this pressure will only continue to increase in the foreseeable future. Complying with “quality tasks” like code reviews takes time away from working on tasks that are directly related to implemented new features. So, the natural tendency is to postpone quality tasks to the stage when features are ready and you have shifted focus to debugging and deployment. This often happens with contractual obligations such as compliance with MISRA coding standards, DO-178B, or similar certifications. For example, assume that after the initial debugging process, the completed code is checked against MISRA standards. Developers then modify the code to try to remove all standards violations. Very often, the process of modifying the code to fix these violations ends up introducing more bugs. This creates the need for yet another debugging cycle, which in turn really degrades productivity and frustrates the entire team.

In the above example, productivity and quality are distinctly at odds with one another. However, this does not have to be the case. You can achieve both productivity and quality by making quality tasks an integrated part of your daily development process.

*continued on page 2*

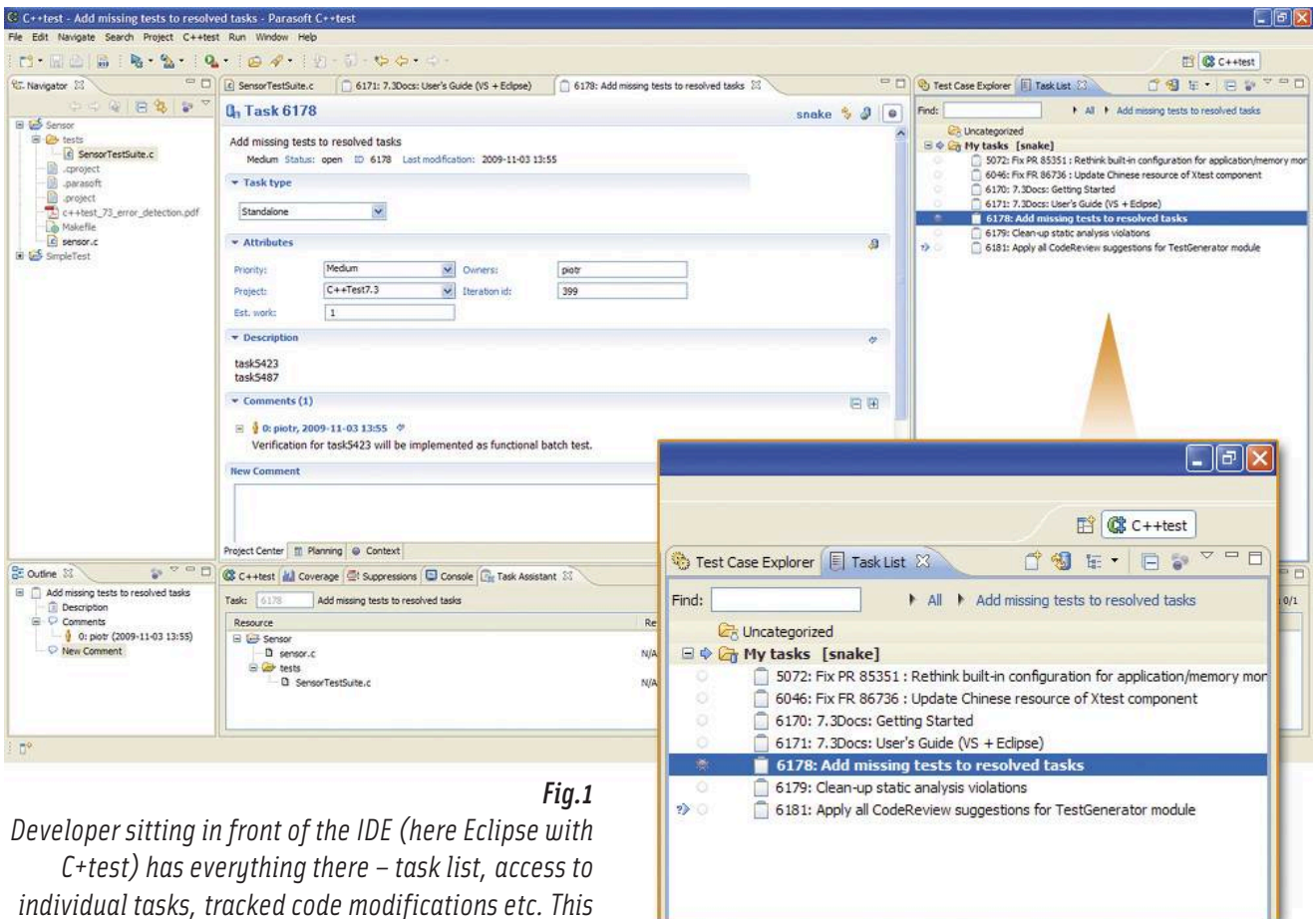
*You can achieve both productivity and quality by making quality tasks an integrated part of your daily development process.*

Here are some rules for building quality tasks into your development process:

1. Quality tasks should be on the same “to do” list as feature implementation tasks (or any other development-related tasks, for that matter). Very often, people underestimate the importance of pulling tasks from disparate sources and consolidating them within the same to do list. Working with multiple lists requires the developer to constantly switch contexts—without guidelines or reminders on when and how to do this. Being human, we will inevitably forget some things—favor the tasks that we enjoy the most.
2. An adequate amount of time must be assigned to those quality tasks (so that fulfilling them does not delay the project). Otherwise, these tasks will be the first thing that is dropped in a rush to meet the deadline.

3. The project (or one of its iterations, if you are taking that approach) must be well-planned, well-tracked and well-managed. A project that constantly slips will be constantly “adjusted”... and quality tasks will likely be one of the first victims of such adjustments.
4. Project leaders need visibility into how a given practice is being followed so that the proper preventive actions can be taken. However, if the other 3 rules are followed, such actions are really very seldom necessary.

For years at Parasoft, we have been providing tools that facilitate best practices and make them easier to apply across a team. C++test is a perfect example of this. It has support for automating code reviews (as much as possible—by nature, code review is inevitably a manual process) as well as providing checkers for coding standards and tools to make unit testing less painful for those who need high line, condition, or MCDC coverage of their test suites.



**Fig.1**  
*Developer sitting in front of the IDE (here Eclipse with C++test) has everything there – task list, access to individual tasks, tracked code modifications etc. This allows for efficient and seamless workflow.*

*continued on page 3*

For just as many years, we have been witnessing the decay of development best practices. Yet once customers finally manage to work through the initial pain, they report how much those best practices actually help them. In fact, we even struggled internally with keeping such quality initiatives alive... until we finally introduced a development workflow based on our latest product, Concerto. This allowed us to comply with the rules above—finally reducing the struggle involved in *actually* doing what we knew we really *should* be doing.

First of all, our developers work off a unified task list that is integrated with their IDEs. All tasks are converted into an integrated “to do” list that may include tasks related to everything from requirements implementation, to bug fixes, to hardware replacement, to code review. This way, team members never have the luxury of

forgetting about quality tasks. Daily, they are reminded to devote time to performing code reviews, fixing coding standards violations, and addressing unit testing or functional testing failures or build warnings. Second, each quality task is assigned an estimated time for completion. Typically, for a 6 week iteration, about 40-48 working hours (5-6 working days per person) are allocated for quality tasks. Although that might seem like overkill, it translates to only slightly more than an hour per day—and it yields significant benefits (which I’ll cover a bit later).

The resulting improved planning gives developers a better chance to actually address their quality tasks. And Concerto itself establishes a central place where the team leader or project manager can monitor how best practices are being followed. [See Fig. 2]

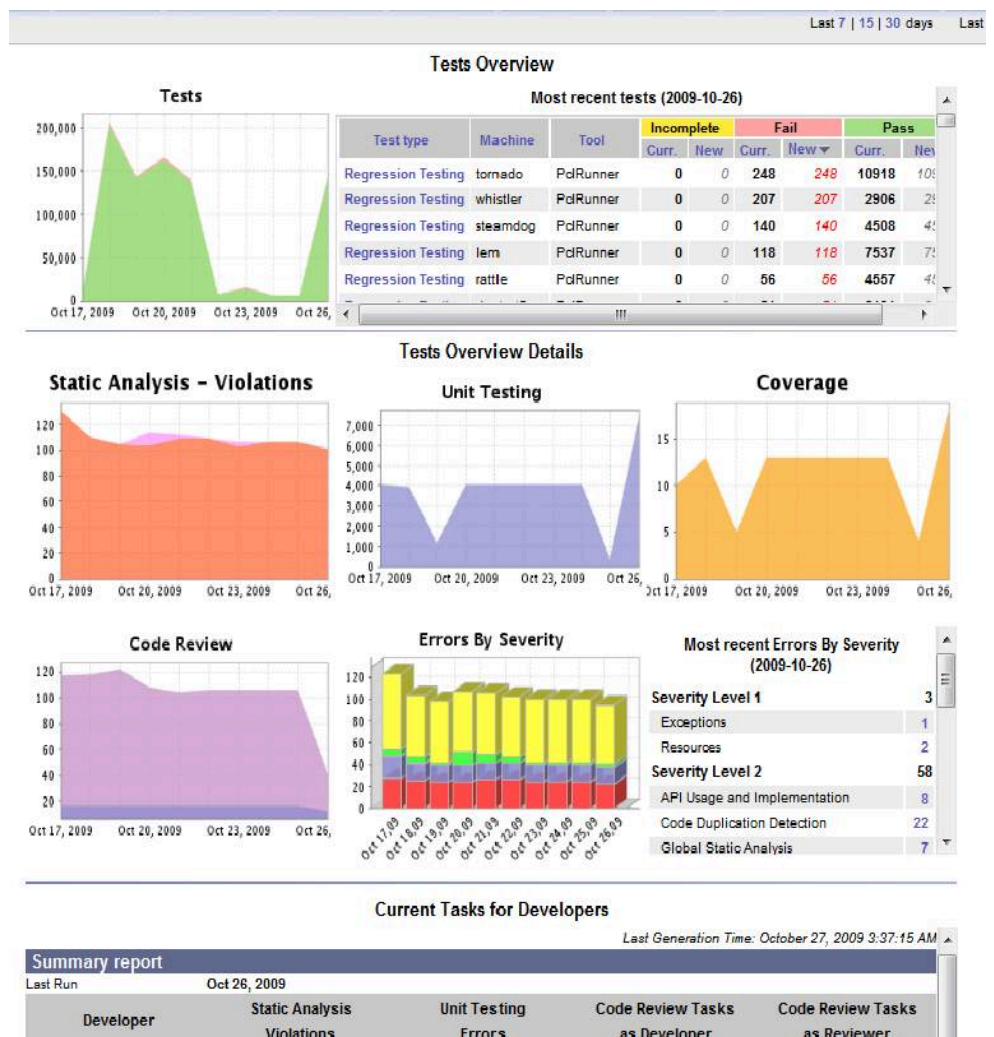


Fig.2

Dashboards like the one on the screen shot allow team leader or project manager to control how team is adhering to the best practices. Here the situation is quite good. Coding standards violations are dropping, regression tests are being executed and code review is a sustainable process.

continued on page 4

*By adopting these rules and applying them to your own process, you can perform quality tasks without suffering from the typical struggle or productivity drain.*

Since introducing Concerto into our development groups in late 2008, we now no longer struggle with keeping our development best practices alive. All our groups do code reviews, build and maintain regression test suits, and clean code against an increasing number of coding standard rules. For example, in one of our projects, we were able to build up to 25 tests per thousand lines of code (KLOC) in a matter of just a few iterations. I'm talking about meaningful functional tests that already proved to find regressions or errors like resource leaks during development. Such regressions are then immediately fixed, all as a part of the quality tasks. Nobody has to be persuaded about how much time this saves. At the same time, the code review process, which is now self-sustainable, usually reveals over 3 issues per person per month. All are logical or design issues, not coding guidelines violations (coding guidelines are checked automatically). Speaking of coding standard violations, within only 3 months, we were able to reduce the number of such violations from 2.3 per KLOC down to 0.2 per KLOC. And coding guidelines cover how you code as well as potentially dangerous constructs or even plain coding errors.

But that's not all. Our team has also become significantly more productive. We release better software in shorter cycles. Mandatory planning forces us to really focus on design upfront, whether we are talking big features or just bug fixes. And the work distribution allows us to assign tasks to the people who are most likely to perform them in the most efficient way. And guess what? The team morale increased because everybody is proud of what they are doing and how they are doing it.

Moreover, records of all the quality practices and implemented features are stored in Concerto. We can analyze the historical results for past iterations and projects to address the issue presented at the beginning of this article: can productivity and quality co-exist? This would involve looking at historical data on items such as:

- How much time have we spent on quality tasks and how much on implementing new features?
- How many bugs have been found in our product later?
- How much time did we have to spend on bug fixing later?
- Are there fewer defects found in our product if we have more automated tests and spend more time on quality tasks?

---

Although initially built for internal use, Concerto became a full-featured product that is currently available on the market. We firmly believe that it can help you to obtain similar results to those that we ourselves achieved—or maybe even better results, especially in combination with C++test. In any case, the rules presented in this article are product-agnostic: they can be applied with any products or solutions. By adopting these rules and applying them to your own process, you can perform quality tasks without suffering from the typical struggle or productivity drain.

**[www.parasoft-embedded.com](http://www.parasoft-embedded.com)**

**Contact info:**

Parasoft Corporation, 101 E. Huntington Dr., 2nd Flr., Monrovia, CA 91016

Ph: (888)305.0041, Fax: (626)256.6884, Email: [info@parasoft-embedded.com](mailto:info@parasoft-embedded.com)