Increase Software Development Productivity:

Equations for Efficiency

By Adam Kolawa, Parasoft Co-Founder and CEO

Why Productivity Matters

In today's economy, software development is a great expense for most organizations, so lagging software development productivity can have a significant impact on the organization's ability to compete and survive. Currently, most software development organizations are not optimized. The C-level still considers software development a cost center with poorly-understood processes and deliverables. There is an increasing demand for software—especially for embedded systems. However, without improved efficiency, it will be difficult to take advantage of these opportunities in a cost-effective manner.

This requires more than tools; it calls for a process that ensures quality software can be produced consistently and efficiently. Like the various automobile manufacturers, different development organizations today typically have access to roughly the same production tools and technologies. The organizations that have a process for leveraging them most successfully are the ones with the highest productivity and the lowest production costs—and the best poised to compete.

This paper presents one simple equation for assessing your team's current efficiency level, then four equations you can apply to improve it:

- Having a clear, actionable plan of attack = Productivity
- Increasing code knowledge = Productivity
- Reducing rework = Productivity
- Reducing debugging = Productivity

How to Calculate Software Development Productivity

There have been many attempts to define metrics that effectively measure software development productivity. Most of the ones that I have seen are amazingly complicated and very difficult to apply.

A simpler productivity metric is the total number of lines of code in the organization divided by the number of people who are working on that code (including QA as well as development). For short, I will call this metric the average LOC per head. Essentially, it represents the developer horizon: how much code each developer really understands.

This measurement is an excellent representation of the development organization's true productivity. If the number is high, it means that you have a relatively low number of people working on code. In other words, you are accomplishing a lot with minimal resources.

Approximately 10-20 thousand lines of code per developer is the norm for most of today's development organizations. This is based on the current industry averages that report a typical program has about a million lines of code, and there are usually about 50 to 80 developers working on that code base.

LOC per head – Does it really vary over time?

After I introduce this metric, I'm often asked, "Does this refer to the number of lines of code per head per month? Per week? Or something else?"

I respond that the time period is irrelevant. Why? Because as a software product evolves, its number of lines of code remains relatively constant.

Among non-developers, there is a misconception that the code base continuously grows as development occurs. However, if you talk to developers, you'll find that most development work—even adding new functionality—really boils down to refactoring existing code.

With the exception of the prototyping phase at the beginning of a completely new development project (a phase that usually lasts a matter of weeks), development primarily involves reworking existing code. Thus, each developer's understanding of the code base is critical for development productivity. As I'll discuss in more detail later, the more code each team member truly understands and can efficiently modify, the more productive the team becomes.

Why Adding Developers Is Not the Solution

It's important to note that adding developers to the project— which is probably the most common attempt to move a project along faster—actually *decreases* productivity according to this equation.

This might seem illogical at first, but when you think about what happens when new developers join a team, it makes sense.

Adding new developers initially hinders the team from making the usual progress on their tasks. First off, existing team members need to spend time training and communicating with the new team members. More critically, new developers have a limited understanding of your team's code base. As a result, they cannot efficiently revise the existing code (which, as I mentioned above, accounts for the bulk of software development), and this limits the team's progress. Their lack of familiarity with the code base also increases the risk of introducing defects and other issues that will later require rework. This too hinders productivity. (In the next section, we'll look more closely at why all these things impact productivity).

Rather than expanding the team size, the key to completing work faster is making the existing development process more efficient: in other words, increasing the average LOC per head. This is the same lesson that was learned time and time again in other industries. As production processes are optimized and take advantage of the latest technologies, fewer people are needed on the "production line" because those people are able to accomplish more than they did previously—in less time.

The next section introduces four ways to optimize the development "production line" to significantly increase the productivity of each developer.

How to Increase Software Development Productivity

Over 21 years of developing software ourselves and assisting other organizations to deliver better software faster, Parasoft has determined that improving software development productivity really boils down to the following equations:

- Having a clear, actionable plan of attack = Productivity
- Increasing code knowledge = Productivity
- Reducing rework = Productivity
- Reducing debugging = Productivity

In fact, these ideas are not so novel. Other industries have struggled with the same types of productivity issues that the software development industry is facing today, and enjoyed tremendous success by adopting the same core approaches. By applying them in our own industry, we can bring software development productivity to more rewarding and competitive levels.

In our experience, we have found that applying the following equations leads to astonishing increases in productivity. For instance, applying them internally at Parasoft enables us to have 120 developers working on 48 million lines of code. This works out to about 400,000 LOC per developer, which is significantly more than the "10,000-20,000 LOC per developer" industry average referenced above.

Equation 1: Have a Clear, Actionable Plan of Attack

The more time a developer spends wondering which requirement or task to tackle next and how to address it, the less time he is actually performing productive, rewarding work—creative tasks that help the organization achieve its goals.

To keep developers focused on performing productive work, define tasks properly, have a system for distributing tasks, and assign tasks properly.

Define tasks properly

In software development, tasks are often quite nebulous: they are vaguely defined, and thus it is difficult for developers to determine where to start tackling them and how to proceed. To overcome this challenge, it's essential to have someone (e.g., an architect or manager) with a good conceptual grasp of the application being developed. This person can take high-level requirements and translate them into more granular tasks that a developer can quickly understand and then successfully implement.

When developers are assigned to work on smaller tasks (for example, work tasks that are scoped to be no greater than one day) as opposed to being assigned to participate in one large, nebulous task, experience has shown that they become much more productive. Since the developers understand exactly where to start and what to do, they become much more efficient: their tasks are attainable and progress is immediately recognized.

Have a system for distributing tasks

Ideally, developers should always have a prioritized list of tasks in front of them, in their natural development environment. This reduces any guesswork as to what is expected and how to proceed.

Assign tasks properly

If a task is assigned to a developer who is unfamiliar with the related piece of code, the implementation time (and the risk of debugging and rework) will be significantly greater than if that same task were assigned to a developer who was intimately familiar with the related code.

Equation 2: Increase Code Knowledge

The speed at which a developer writes code has nothing to do with how fast the developer types. It's all about how well he understands the code that he is asked to modify. When a developer is assigned a task, he needs to understand what the existing code is doing, how to change what it's doing, and be able to foresee the impact of his changes. This requires code knowledge. Thus, expanding each developer's code knowledge will broaden the range of tasks that you can assign each developer, as well as reduce the time required to complete each task. How do you achieve this?

One way is through peer code review. When one developer reviews another's code, he learns about code complexities and connections. This ends up expanding the amount of code that each developer can work on, and the speed at which tasks are completed. In addition to expanding the knowledge of the reviewers, this process also provides additional insight to the reviewee, who receives valuable team member feedback on how the code he wrote relates to the code base as a whole.

Another way to increase code knowledge is through regression testing. If a developer is not familiar with the code and has to change it, his greatest concern is probably breaking something that's already working. Regression test suites instantly alert developers to any unexpected impacts—for instance, if a modification changes or breaks some existing functionality in another part of the application. In addition to teaching developers them about correlations between the modified code and the other parts of the code base, this also increases productivity by giving developers the courage to improve and extend code without being stunted by fear.

Expanding each developer's code knowledge also promotes productivity by preventing mistakes that lead to debugging and rework, which are discussed in the next two equations. The less familiar developers are with the code, the more prone they are to make mistakes that will slow productivity long after the implemented piece of code is first checked in.

Equation 3: Minimize Debugging

Debugging is essentially the process of trying to fix what was implemented incorrectly. The more time developers spend fixing code, the less time they have to spend on more productive tasks. One way to minimize debugging is to establish an infrastructure that prevents defects from entering the code base. This could include automated defect prevention practices such as static analysis, unit testing, and so on.

The true value of minimizing debugging through defect prevention is that it reduces the need for rework. In the long run, performing defect prevention practices that prevent entire classes of errors as code is being written is considerably more efficient than constantly chasing after defects one by one and then fixing each occurrence later in the development process—when it is more difficult, costly, and time-consuming to do so.

Equation 4: Minimize Rework

Minimizing debugging is just one way to minimize rework. Other ways are related to requirements.

If architects and managers do not understand requirements from the start and, in turn, developers do not implement them correctly, the team will inevitably end up wasting tremendous amounts of time reworking the code. It is critical to implement requirements as the customer expects and the specification states. To achieve this, you need to ensure that each party truly understands the requirement before taking it to the next step.

To understand what this involves, consider this analogy: A developer's work is much like that of a building contractor. First, the architect finds out what the customer wants and sets the requirements. From there, just as contractors implement a building plan, developers start writing code.

At this point, mistakes can stem from two key sources:

- The architect misunderstands what the customer wants, which leads to incorrect requirement specifications.
- The developer misunderstands requirement specifications and implements them incorrectly.

Prototypes are very effective at preventing rework related to architect-customer misunderstandings. A prototype gives customers a chance to visually review what the developers are building for them and immediately alert the team if the project seems to be heading in the wrong direction.

Developer-architect misunderstandings can often be flushed out through a policy that as developers write code for a new requirement, they also need to write a test case that verifies if the requirement is implemented correctly. This forces developers to look at requirements from two different angles—implementation and validation—which results in a thorough understanding of each requirement

Productivity = Quality

This is obviously not an exclusive list of productivity equations, but rather an overview of those that we've seen have the greatest impact at our own organization as well as other organizations.

Surprisingly, the impact of applying these equations extends beyond productivity into quality. Quality and productivity are inextricably intertwined. If you follow these equations, your application quality will increase because you will have established a much less error-prone development process as a result of having:

- Developers who truly understand the code base they are working with and the implementation tasks they are asked to perform.
- An infrastructure for preventing many errors that would otherwise require debugging and rework.

In the end, a process that results in low-quality software can never be productive. Rework and debugging will undermine any gains made by delaying or discounting the role that quality plays in fostering long-term productivity.

Increasing Productivity with Parasoft Concerto

Parasoft Concerto helps organizations deliver quality software consistently and efficiently. It seamless integrates into any development environment, working in the background to help you:

- Convert high-level requirements into actionable, measurable tasks.
- Distribute and manage tasks across development and QA.
- Drive tasks to completion according to a policy-driven quality workflow.
- Monitor implementation quality, status, and compliance objectively, in real time.

The end result is dramatically increased process visibility and control. For instance, managers can instantly verify whether a project is on time and on budget. They gain an objective, unobtrusive way to check whether the expected quality is achieved and the designated policies are followed.

Following is a quick overview of how Parasoft Concerto helps organizations apply the equations outlined in this paper:

Equation	Parasoft Concerto Support
Having a Clear, Actionable Plan of Attack=Productivity	• Helps managers dissect high-level requests into reasonable and manageable tasks. • Allows managers to create and assign tasks from a central console. • Distributes tasks directly to the developer's IDE based on manual assignments or business rules. • Helps managers estimate the available resources and allocate tasks accordingly. • Development progress is measurable and the implementation status of project artifacts (requirements, tasks, defects/ enhancements) is visible to QA so they know what's ready for testing.
Increasing Code Knowledge= Productivity	• Enforces policies for increasing code knowledge (peer reviews, etc.). • Automates the peer code review process to expose developers to more code. • Automates the regression testing process to ensure that code impacts are exposed immediately.
Minimizing Debugging= Productivity	• Integrates quality into the SDLC to ensure that quality and error remediation tasks are accomplished according to the organization's prescribed policies. • Automates a complete spectrum of defect prevention and quality assurance practices across the SDLC, including: o Pattern Based Static Code Analysis o Flow Based Static Code Analysis o Code Analysis for Security o Code Review o Contextual Code Review o Unit Testing o Memory Error Detection o Message/Protocol Testing o Web UI Testing o End-to-end Testing o Functional Testing o Load Testing o Change-based Testing o Manual Testing o User Acceptance Testing
Minimizing Rework= Productivity	• Connects to requirements management systems; drives an automated workflow for collecting and evaluating requirements. • Prevents customer-architect misunderstandings by supporting Agile development processes with prototyping, short iterations, and other practices that promote early and frequent customer interaction. • Prevents architect-developer misunderstandings by enforcing policies such as requiring that a test case be written for every use case—forcing developers to think about each requirement from different perspectives.

Learning More

To get more details about how Parasoft Concerto helps organizations achieve these objectives, read Parasoft's "Software Development Management" white paper.

About Parasoft

For 21 years, Parasoft has investigated how and why software defects are introduced into applications. Our solutions leverage this research to dramatically improve SDLC productivity and application quality. Through an optimal combination of quality tools, configurable workflow, and automated infrastructure, Parasoft seamlessly integrates into your development environment to drive SDLC tasks to a predictable outcome. Whether you are delivering code, evolving and integrating business systems, or improving business processes—draw on our expertise and award-winning products to ensure that quality software can be delivered consistently and efficiently. For more information, visit http://www.parasoft.com.

Contacting Parasoft

USA
101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe
France: Tel: +33 (1) 64 89 26 00
UK: Tel: + 44 (0)208 263 6005
Germany: Tel: +49 731 880309-0
Email: info-europe@parasoft.com

Asia
Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

Other Locations
See http://www.parasoft.com/contacts