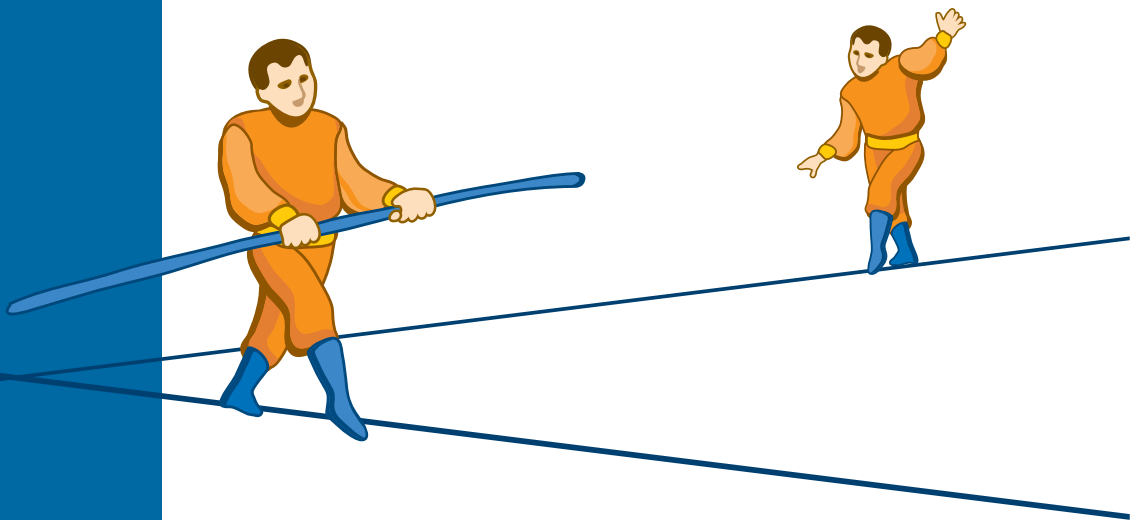


JetBRAINS

 **IntelliJIDEA**
The most intelligent Java IDE around



**Maximizing stability
with IntelliJ IDEA**

CONTENTS

Introduction: to IDE or not to IDE?	2
Why IntelliJ IDEA	3
Higher Comfort and Productivity	3
Details make the difference	3
When an editor is really intelligent	3
IntelliJ IDEA: Higher ROI	4
Coming In On Time and Under Budget	4
Static Analysis – Effective Remedy	5
Refactoring – Reduce Maintenance Costs	6
IntelliJ IDEA – How You Can Sell Better?	7
Conclusion	9
Develop with pleasure	10
Customers	10
Awards	10
References	11

Introduction: to IDE or not to IDE?

“There are two kinds of people in the world, those who believe there are two kinds of people in the world and those who don’t.”

Robert Benchley

*Benchley’s Law of Distinction
US actor, editor, author & humorist (1889 - 1945)*

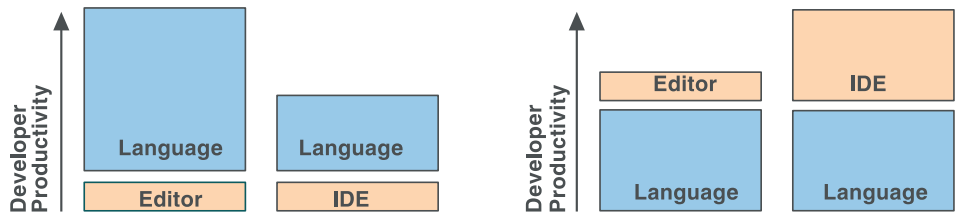
According to research completed by the Standish Group [7], only one third of U.S. projects in 2004 were completed on time and on budget, with all features and functions originally specified. More than half were completed over budget, late, and with fewer features and functions than initially specified. Each sixth to seventh project failed entirely.

To mitigate these risks and succeed, one needs the means to facilitate the development process, improve code quality and lower the total project cost of ownership and related expenses. And quite clearly, at the core of any software project are its developers and the tools they use.

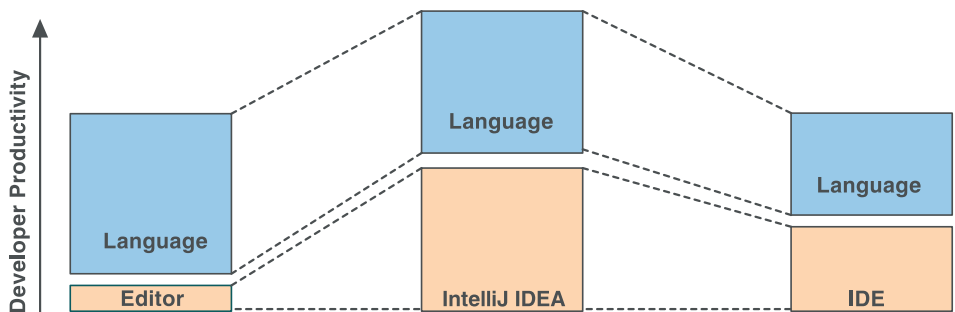
The “true believers” of the software development world actually follow two opposite “religions”:

- The first camp believes: *The more experienced your team, the better the results*
- The second one states: *The better your development tools, the more productive your team.*

This idea was nicely explained to the public by Oliver Steele [1], who has even created a graphical representation of these two points of view:



This picture very closely emulates reality, but the question arises: *Why are people so rarely ready to compromise?* Why don’t they believe that combining the two approaches can make the productivity picture much more attractive? Just equip a development team of any level with a professional tool like IntelliJ IDEA and the result will be immediate and compelling:



Why IntelliJ IDEA

Higher Comfort and Productivity

“IntelliJ IDEA has enabled us to use a compact, extremely usable development environment for all our developers without any additional installation or customization efforts. By bringing good ideas and patterns near to our developers, we have enabled them to ‘learn by doing,’ having overall quite impressive results.”

Christopher Semturs
 Product Development
 OBJENTIS Software
 Integration GmbH

“IntelliJ IDEA has a very convenient and user friendly interface and a lot of handy and helpful features. Its ‘IntelliJence,’ look and feel, and speed compared to the rest of crowd, especially with IntelliJ 4.0, have been a big plus.”

Chetan Shah
 Executive Vice President,
 Technology
 Synogy, Inc.

“IDEA ‘knows’ everything you want to do. The product boosts your productivity immediately and after that, the more you put into it, the more you get out of it.”

Ken Watts
 VP Platform Engineering
 Documentum

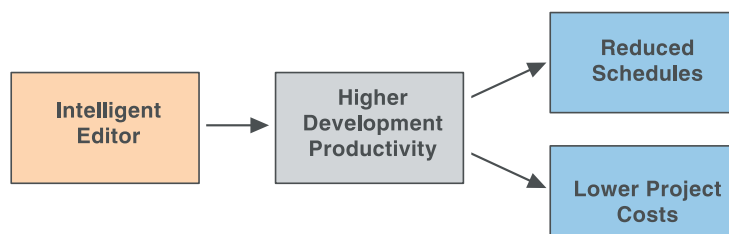
Details make the difference

When speaking about how comfortable developers feel with IntelliJ IDEA, we encounter a serious problem – comfort cannot be measured. How can one explain in measurable terms, for example, why one chair is more comfortable for you than another? What feature or quality is more important than another, and TO WHAT EXTENT? Certainly, it is possible to compare each feature step by step – this back is more ergonomic, that arm is adjustable, the swinging action has an adjustable spring, this has an adjustable head-rest, this is... It would be great if such advantages were few, easily identified and easily measured. But what if there are hundreds? Or thousands? What if the comfort level results from many small details – what then?

IntelliJ IDEA’s comfort level is something that can be really measured empirically. And it is immediately evident. IntelliJ IDEA simply never gets in the way. It never imposes a way of doing things. Rather, it unobtrusively provides intelligent assistance and suggests improvements like no other development tool.

When an editor is really intelligent

Each of the individual features in IntelliJ IDEA increases the user’s comfort and contributes to productivity, the way closer tolerances and finer construction in a musical instrument pull the best performance from a musician. If “the devil is in the details,” so is the intelligence that makes a Best of Breed IDE. For example, IntelliJ IDEA’s savvy code completions, convenient-to-use intention actions, unparalleled code analysis tools and many incremental editor enhancements – each feature separately is top-notch in its class. But IntelliJ IDEA’s true strength and unsurpassed advantage are in the whole. Being fused together, its features form an orchestra of powerful instruments. IntelliJ IDEA makes the developers really creative and pulls the very best from them during development.



What benefits will any team gain by using an intelligent editor like the one built into IntelliJ IDEA? Increased comfort and enhanced development productivity, at minimum. The result will be immediate and visible, as lower project costs and accelerated time-to-completion.

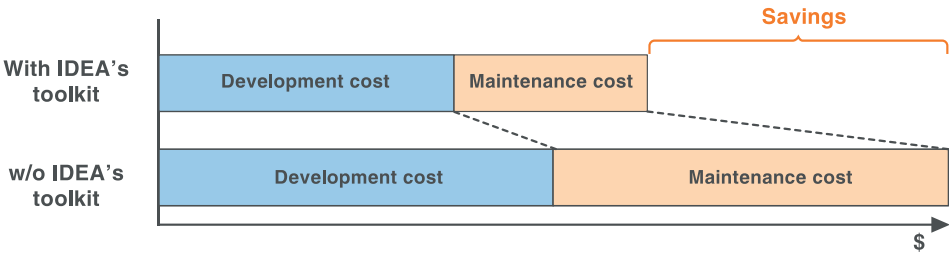
IntelliJ IDEA: Higher ROI

“IntelliJ IDEA will save you time and money. For developers, IDEA promotes the rapid creation of clean, reusable, and well documented code. For managers and architects, IDEA makes it possible to quickly navigate, review, and refactor code projects large and small.”

Mike Sick

VP Development
UserMagnet, Inc.

Besides comfort, there is another benefit of IntelliJ IDEA – and it can be measured. That is how cost efficient it is to own IntelliJ IDEA.



At first glance, it may not be obvious how a commercial product can allow a company to save money compared to using free products. As Boehm and Sullivan noted in their analysis of modern software economics [5], most IT decisions in the industry today are made by finding the minimum cost approach as opposed to searching for the maximum value solution. It takes more than remarking, “You get what you pay for,” to overcome this systemic predisposition to value short-term pennies above longer-term dollars.

“From a business point of view, the cost is very attractive. Not free, but at a really good price point that immediately offers a very apparent ROI.”

Bruce Kratz

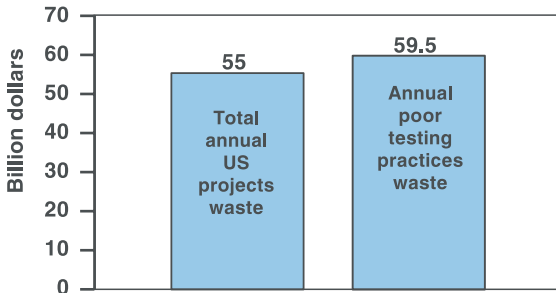
Director, Engineering East
HP Web Services Management
Organization

Let us take a look at the state of projects in the software industry to see why those tempting short-term pennies often turn into surprisingly expensive drains on productivity and operating budgets.

Coming In On Time and Under Budget

Detailed project analysis shows [6] that for a single software organization, **the estimated time to rework code (fix defects and bugs in software) can be as high as 80% of total project development costs.** Does anybody plan for such an incredible expense in their project budgets to rectify mistakes? Of course not, but mistakes do come to pass nevertheless.

- The 2004 CHAOS report found total U.S. project waste to be about \$55 billion [8] (including lost dollar value and cost overruns). Total project spending was found to be \$255 billion in the 2004 report.
- The U.S. Department of Commerce [4] reported that the economic consequences of inadequate software quality management practices, namely testing, amounted to \$59.5 billion wasted per year.



The numbers are similar, aren't they?

It is obvious that mistakes in the testing process can give rise to monstrous results – all the way up to total project failure.

As originally noted by Barry Boehm, the **defect repair cost increases by a factor of ten for each successive stage** of development. A defect that costs one hour to find, fix, and retest in unit tests, would cost ten hours in the next stage, 100 hours two stages later (system test) and 1000 hours after shipping.

With such exponential increases, we should hardly wonder at the astronomical costs of overruns!

Fortunately, Software Engineering has in its arsenal various efficient methods to dramatically reduce such noticeable overhead. IntelliJ IDEA contains many cost reduction features that come standard. Furthermore, the makers of IntelliJ IDEA were the first to realize the importance of such features, implement them in a product, and in the process create what is consistently referred to as the industry’s leading Java IDE.

Static Analysis – Effective Remedy

“Editor’s ability to... perform background code analysis... allows developers to write code that compiles and runs correctly the first time, reducing the time-consuming edit-compile-debug cycle.”

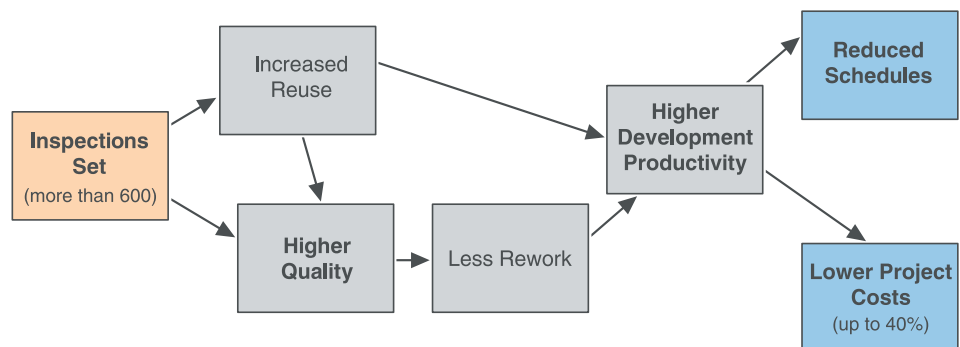
Gordon Tyler
Software developer
Quest Software, Inc.

Static analysis (code inspections) is a set of techniques for analyzing source code and software designs. It does not require the software to be executed to produce results. As Khaled El Emam shows [2, 3], the average effectiveness of inspections (in proportion to defects in the code that are found during inspection) is 57%. This means **more than half of the defects in the code can be found using inspections**. According to Khaled El Emam, **code inspections allow one to reduce any project cost by 35-40%** (and also raise code quality and reduce risk of project failure) due to timely location and correction of defects, increased code reuse, and higher development productivity.

The figure below shows the sequence of mechanisms that would lead to concrete benefits from the use of static analysis features, as found in IntelliJ IDEA.

“Most of our engineers do not get any errors at all during compiling because IDEA catches them all during editing. The time we save is remarkable.”

Ken Watts
VP Platform Engineering
Documentum



IntelliJ IDEA is the only IDE with *more than 600 types* of intelligent automated code inspections integrated seamlessly within it. Many competitors don’t even have inspection functionalities.

Besides finding defects, IntelliJ IDEA eases the problem-correction process by

providing ready-to-go improvements for the most frequent errors. Moreover, a powerful on- the-fly code analysis mechanism makes code analysis even more efficient, showing possible defects instantly. As a result, the code is less error-prone; so there is no need to recompile it frequently, which saves additional time, especially on large projects.

Static analysis results in higher code quality. As we will show below, higher product quality also makes for increased profit.

“IntelliJ IDEA’s powerful refactorings coupled with its deep insight into our code provide us with the knowledge and the tools to write new code and maintain existing code in an efficient manner.”

Gordon Tyler

Software developer
Quest Software, Inc.

“IntelliJ IDEA helps us by providing assistance in the every-day task of writing code by automating repetitive tasks in an intelligent manner and giving incredible insight into our code’s structure and semantics.”

Gordon Tyler

Software developer
Quest Software, Inc.

“IDEA is the best tool to use to learn about code that you didn’t write personally.”

Brian Robinson

Senior software designer
HP Web Services Management
Organization

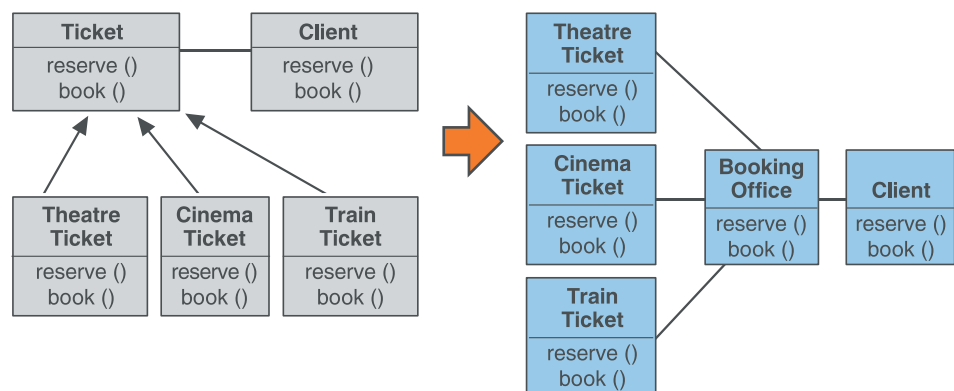
Refactoring – Reduce Maintenance Costs

Current IT market research [9, 10, 11] shows that **customization, including legacy modernization and integration projects, represents more than half of all software budgets**. Average maintenance costs are higher than development costs, and this excess – especially for custom system development – can turn out to be a multiplier of two, ten, or even one hundredfold!

Research by Robert M. Leitch [12, 13] showed that **applying refactorings can reduce software maintenance costs by almost half**.

Refactoring is not some esoteric programming technique conceived just for computer science students to argue about with their professors. The problems it solves are expensive and often have strong, negative consequences for companies in terms of customer relations, data accuracy, downtime, and code maintenance. Properly carried out, refactoring makes code less expensive to maintain, easier to update, more reusable for later projects, and less confusing to other programmers either in other departments or who come along later to reuse or redesign it. For companies, the qualitative ROI is immediate as bugs stop creeping into code, clean code is produced faster, and fewer programmers have to spend time correcting the mistakes of others. **These savings quickly become quantitative, as general overall costs go down.**

Refactoring enhances architectural flexibility. Refactored code is reduced in overall size, so it becomes less knotty and complex, and becomes easier to read. Such code can be more easily modified, extended and adapted without risking complete system disintegration.

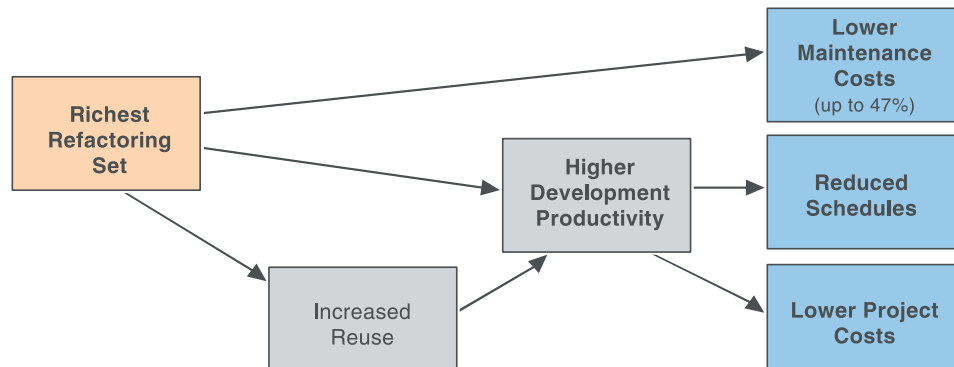


For everyday code refactoring, IntelliJ IDEA can become a very powerful helping hand for any mature developer, saving invaluable time and resources (savings which equate to lower project costs).

In this area, among others, IntelliJ IDEA excels. It offers an extensive refactoring arsenal. More than 60 distinct tools automate time consuming and tedious refactoring procedures and make IntelliJ IDEA the industry's leading refactoring specialist. **In refactoring, IntelliJ IDEA has no equal.**

"IDEA's refactorings give our developers the ability to make structural changes to their code with the confidence that it will still compile and run correctly afterwards."

Gordon Tyler
Software developer
Quest Software, Inc.



Coding with a tool like IntelliJ IDEA is safer; it does not let you produce incompatible code, and always provides the necessary warnings to help avoid such an outcome. Moreover, its powerful UNDO stack, plus smart Local History feature, lets you roll-back to a previous file version, taking into account all relevant project-level changes. All IntelliJ IDEA components work as a single, well-tuned machine, in total harmony, a performance level impossible to obtain from any IDE with loosely coupled modules.

"IntelliJ IDEA improves both the quality and predictability of schedules, which improves customer satisfaction, so it has a positive effect on the bottom line."

Ken Watts
VP Platform Engineering
Documentum

IntelliJ IDEA – How You Can Sell Better?

It is well known that the higher quality code a software company produces, the lower a customer's post-purchase costs. But what isn't as well known, is how much one can also save in the process of producing better code.

Based on data collected about software defect density from companies at different maturity levels and published by C. Jones [14], and using his own method, Khaled El Emam in his survey has calculated the reduction in customer ownership costs as a result of a vendor's improved software quality.

It was found that the cost to customers lowered dramatically as supplier and software maturity increased. **A customer's cost reduction between the worst/average and best-in-class companies, depending on business sector, project size, and country, ranged from 53% to 74%!**

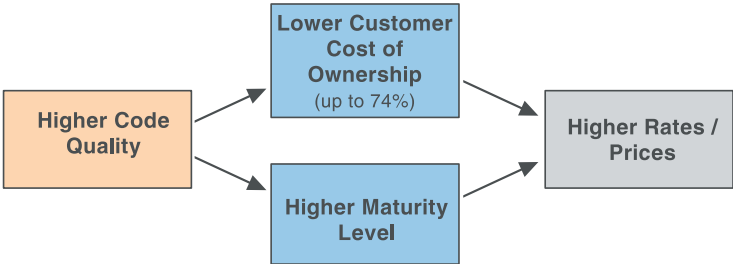
This means a customer who buys software from a company that delivers a best-in-class product **cuts post-purchase costs by about 70%** compared to those who buy the same type of software from a worst-in-class performer, and **more than 50%** less than if they buy only an average performer.

“We’re developing enterprise-level software for financial institutions, telecoms, and other large customers who spend anywhere from hundreds of thousands to millions of dollars on our products. Fault tolerance, high scalability and non-stop operation are critical and our code has to be top quality. IDEA allows us to produce a better product, faster and more efficiently.”

Bruce Kratz

*Director, Engineering East
HP Web Services Management
Organization*

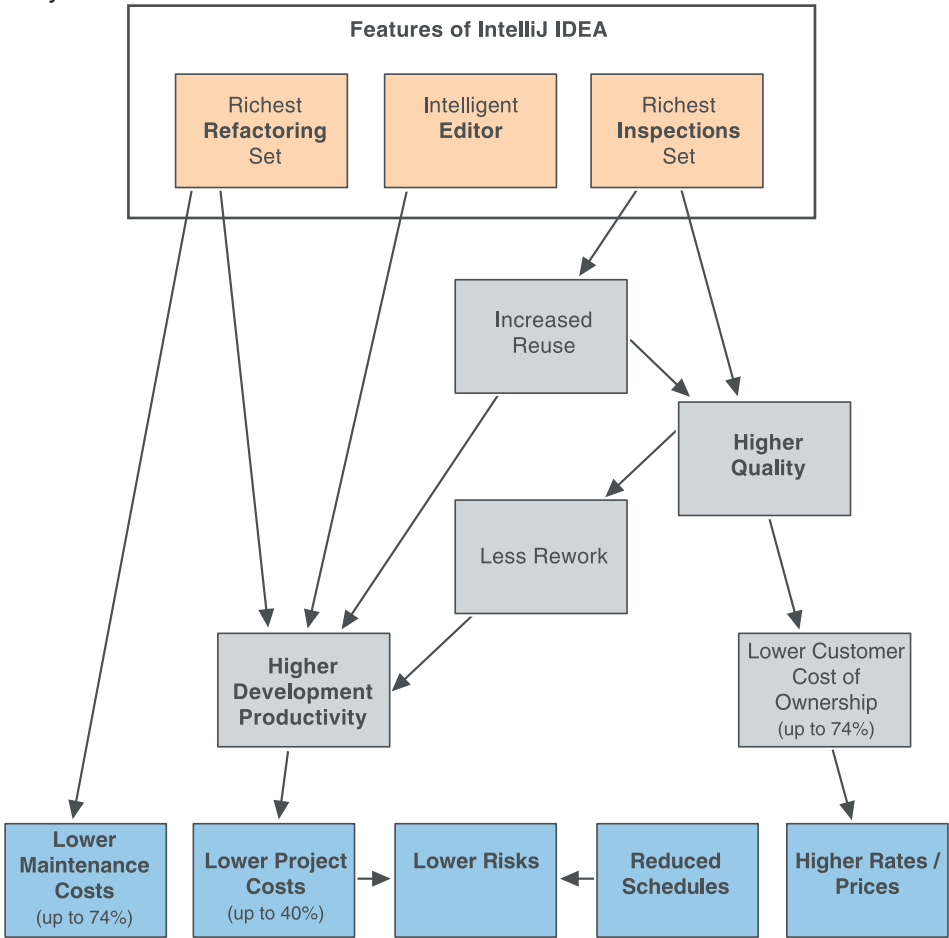
The fact is, companies that produce higher quality software look more attractive to the market, and not only that, they can better justify their prices and rates, because their code quality provides an overall savings to their customers – savings that, in many cases exceed the actual initial and annual licensing costs of the software.



All these factors hold true for IntelliJ IDEA. JetBrains developers write the company’s products using an advanced intelligent development tool (IntelliJ IDEA) so they produce higher quality code. JetBrains customers who use IntelliJ IDEA save on their postpurchase costs, just as do their customers.

Conclusion

In the end, what does it all add up to? On the one hand, by using IntelliJ IDEA you can dramatically reduce your project expenses and maintenance costs. On the other, you can better justify your service rates and product prices because you produce best-of-breed code. Clearly, your revenue will shoot up. You'll see the results as soon as you start using IntelliJ IDEA, and you have nothing to lose but your old IDE.



Higher developer productivity follows from improved working comfort. More defects are detected and corrected before they affect the overall project. The result is higher code quality, a shorter time-to-market period, and additional dividends from having a good reputation.

Can adopting the best-of-breed IDE produce all these benefits? YES. It's an easy assertion to prove – simply use IntelliJ IDEA, experience its benefits, and you'll never look back.

Develop with pleasure

Customers

More than 3000 companies, including more than 100 of the top Fortune 500 firms, spanning over 70 countries, now put their trust in JetBrains. They represent such diverse industries as finance & banking, automotive, biotechnology, enterprise system software development, IT infrastructure manufactures, mobile handset makers, fossil fuel & petroleum energy, and much more. Additionally, JetBrains tools are used by universities all over the world to help simplify teaching and to encourage improved methodologies and practices in object-oriented programming.

Fortune 100

American Express
 Bank Of America
 Cisco Systems
 Citigroup
 FORD
 FREDDIE MAC/MERRIFIELD WHSE
 Hewlett-Packard Company
 GE Medical Systems
 Lockheed Martin
 Merrill Lynch
 Northrop Grumman
 Verizon
 Walt Disney World
 Wells Fargo Bank

Fortune 500

Agilent Technologies, Inc.
 Allegheny Energy Supply
 Conseco
 DTE Energy
 EMC
 Fannie Mae
 First Data Corporation
 Humana
 Morgan Stanley Capital International S.A.
 NIKE Inc.
 Oracle
 Raytheon Company
 Sun Microsystems
 Tribune Interactive
 Xerox Corporation

Awards

 2008, 2005, 2004	 2007, 2006, 2004, 2003	 2005	 2005	
 2004, 2006	 2004, 2003, 2002	 2003, 2002	 2003	 2002

References

1. Oliver Steele, "The IDE Divide", <http://osteele.com/archives/2004/11/ides>
2. Khaled El Emam : "Return on Investment Models for Static Analysis Tools", Klocwork., 2003
3. Khaled El Emam : "The ROI from Software Quality: An Executive Briefing," K Sharp Technology Inc., 2003.
4. RTI, "The Economic Impacts of Inadequate Infrastructure for Software Testing," Technical Report, National Institute of Standards and Technology 2002.
5. B. Boehm and K.J. Sullivan, invited paper, "Software Economics: A Roadmap," 22nd International Conference on Software Engineering, June, 2000.
6. Shull, F., Basili, V. R., Zelkowitz, M. V., Boehm, B., Brown, A. W., Costa, P, Lindvall, M., Port, D., Rus, I.,and Tesoriero, R. "What We Have Learned About Fighting Defects". International Software Metrics Symposium, Ottawa Canada. 2002.
7. Annual CHAOS Report, The Standish Group International, Inc., 2004
8. 8 CHAOS Chronicles, The Standish Group International, Inc., 2004
9. "How IT spending is changing", McKinsey Quarterly, 2004
10. Morgan Stanley CIO Survey, 2003
11. Ian Sommerville, Lutz Prechelt, "Software Evolution", Ian Sommerville 2004, Software Engineering, 7th edition
12. Robert M. Leitch, "Assessing the Maintainability Benefits of Design Restructuring using Dependency Analysis", University of Alberta, 2002
13. Eleni Stroulia, Rob Leitch, "Understanding the Economics of Refactoring", MacDonald, Dettwiler and Associates, Ltd., 2003
14. C. Jones, "Software Assessments, Benchmarks, and Best Practices: Addison-Wesley", 2000.