Borland®

Borland® Kylix™ 3 versus Linux® GCC Development

A productivity and maintainability comparison

by William Roetzheim and John Amacker,
the Cost Xpert Group
for Borland Software Group
September 2002

Contents

Introduction

This paper utilizes parametric modeling techniques to perform side-by-side comparisons of software development using Borland® Kylix™ 3 versus Linux® g++, the name sometimes used for the GNU C Compiler (GCC) when it is being used to compile C++ programs. We examine typical embedded projects, server-side Internet projects, and client-server applications, then look at the relative efficiency of the two development environments in terms of effort, cost, schedule, quality, and life-cycle costs. In all dimensions of this comparison, it is established that Kylix 3 outperforms g++, often with an improvement of two-fold or more.

Background

According to the IDC report "*Worldwide Linux Operating Environments Forecast and Analysis, 2002-2006: A Market in Transition*" (July 2002, Gillen, Kajita, Kusnetzky, Brewer, Hingley, Lee, Manfrediz), "IDC expects spending on Linux operating environments to increase over the next five years from $80 million in 2001 to $280 million in 2006, a 28% compound annual growth rate (CAGR)." In the same report, IDC notes, "Despite the unconventional way Linux is bought and sold, it has become a mainstream choice for many infrastructure workloads particularly because the software is available either freely on the network or as a low-cost packaged product that can be deployed on low-cost, high-volume systems. Furthermore, Linux is often packaged with other open source software such as Samba for file/print services, Apache for Web Services, and

Kylix™

white paper
white paper

MySQL™ or PostgreSQL for data management, which makes it a highly functional and cost-effective environment." Current development in a Linux environment is primarily accomplished using the Linux GCC (C language compiler), often with a front-end that allows developers to work with different higher-level languages. One common approach is to use g++, a C++ front-end to GCC.

GCC also can compile programs written in C, C++, Objective C,® Ada 95, Fortran 77, and Pascal. GCC refers to the compilation system as a whole, and more specifically to the language-independent part of the compiler, which is also known as the back-end. g++ builds object code directly from C++ program source, i.e. there is no intermediate version of C. g++ takes advantage of most of the GCC extension to the ANSI C standard. For more information on GCC and g++ please visit the GNU official site: (www.gnu.ai.mit.edu).

Borland has recently introduced Kylix 3 for Linux development. Kylix 3 supports both C++ and Borland Delphi™ language programming and provides an ideal set of development tools, both for implementation and for testing. In the Windows® platform, Delphi, Borland C++Builder® and similar high-productivity development environments have drastically improved the productivity of software developers, resulting in faster time-to-market and lower cost development. The compatibility of Kylix 3 with C++Builder 6 and Delphi 6 enables cross-platform Linux and Windows development.

All three products include CLX™ (Borland® Component Library for Cross-platform), a library that simplifies and standardizes reuse of code objects. In total, CLX features more than 190 reusable software building blocks that simplify development greatly. Kylix 3 also includes an advanced code editor, CodeInsight,™ a fully integrated graphical debugger, and a 32 bit optimizing C/C++ code compiler. In this paper, we explore productivity rates for Kylix 3 vs. g++ development and quantify typical impacts on development cost and schedule.

Objective

We identified three classes of software development projects that we felt were typical projects applicable to the Linux environment. These were:

- An embedded application;
- A Web server application; and
- A client-server application.

We set out to quantify the difference in developing these three representative applications using Kylix 3 versus g++. For each project, we wished to explore statistically significant differences in the areas of:

1. Effort/cost;
2. Development time;
3. Maintenance effort following deployment; and
4. Documentation needed to be created during development.

Finally, we wished to determine whether the results we discovered would vary significantly in relation to the size of the development project. We therefore repeated our work using three client-server projects that differed only with respect to the software application's size.

Approach

Kylix 3 is too new to have an extensive experience base of metric information from deployed projects. Even for well-established development platforms, data for identical projects developed using different tools is virtually impossible to obtain. Luckily, there is an approach to accomplish our objectives. Cost Xpert (www.costxpert.com) is a software project modeling tool that incorporates more than 70 parametric models to accurately model and predict software project behavior to better than 10% of actual results. Version 3.2 of this tool supports Kylix and g++ projects. By defining the exact same project exclusive of the development tools used, then modeling the project outcomes, it is possible to compare predicted outcomes for these two environments when applied to the same project. Differences greater than 10% may be considered to be statistically significant.

The project definitions used for the initial analysis are described in the following paragraphs.

- All projects were defined based on a development team with three or more years experience in the application domain; 12 months experience with the programming language (either C++ or Object Pascal); at an average hourly rate of $100 per hour. The development was assumed to be performed at a single site with a LAN installed.

- The sample embedded application consists of 2 control classes, 3 interface classes, 10 other classes, 3 data stores, 75 class methods/functions, and 1,500 additional lines of code to handle miscellaneous requirements not otherwise covered.

- The sample Web application consists of 2 file-oriented external interfaces, 20 tables in a database, 10 hard copy management reports, 10 static screens, 10 screens with dynamic update from the database, and 10 interactive screens where the user interacts dynamically with the system. In total, the application consists of 207 Internet Points worth of delivered functionality.

- The sample client-server application consists of 2 file-oriented external interfaces, 20 tables in a database, 10 hard copy management reports, and 20 screens. In total, the application consists of 344 Function Points worth of delivered functionality.

To test the impact of project size on the results, we defined the client-server project described above as a medium sized client-server project. We defined small and large client-server projects as follows:

- The small client-server project consists of 1 file-oriented external interface, 3 tables in a database, 2 hard copy management reports, and 4 screens. We felt that this project would represent a very small project, for example, a small support application. In total, the application consists of 63 Function Points worth of delivered functionality.

- The large client-server project consists of 25 external interfaces, 5 external queries or messages, 200 tables in a database, 40 hard copy reports, and 90 screens. In total, this application consists of 2755 Function Points worth of delivered functionality.

Now, let's look at the results of our analysis.

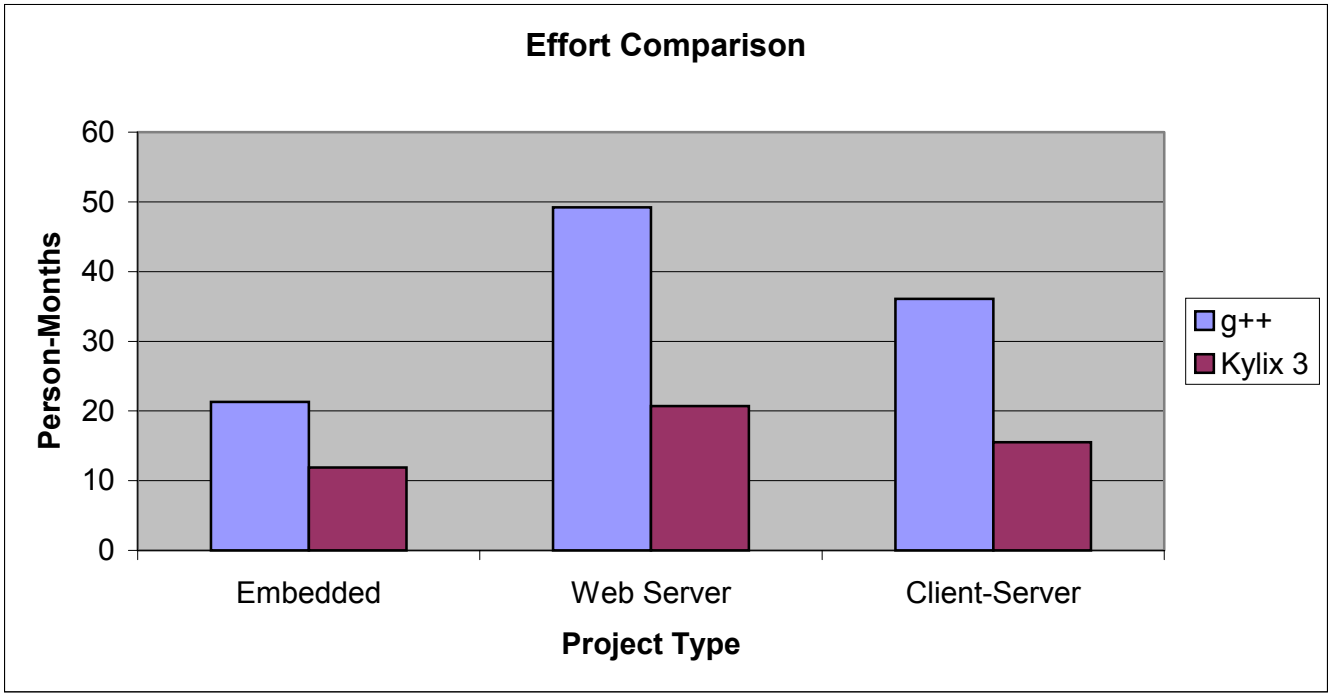
Results

In this section we'll look at our results across each dimension of study:

1. Effort/cost;
2. Development time;
3. Maintenance effort following deployment; and
4. Documentation needed to be created during development.

Effort and cost

The following figure shows the results of our effort analysis for building the three categories of systems using Kylix 3 versus g++. In all three cases, Kylix 3 resulted in a substantial reduction in development effort. For the Web and client-server applications, the Kylix 3 effort was about 42% of the effort required for an equivalent deployment using g++. Even in an embedded environment where g++ is presumably better suited, the Kylix 3 development could be accomplished in roughly half the effort required for the g++ environment.



Of course, the costs follow a similar pattern as shown in the following table.

Cost comparison

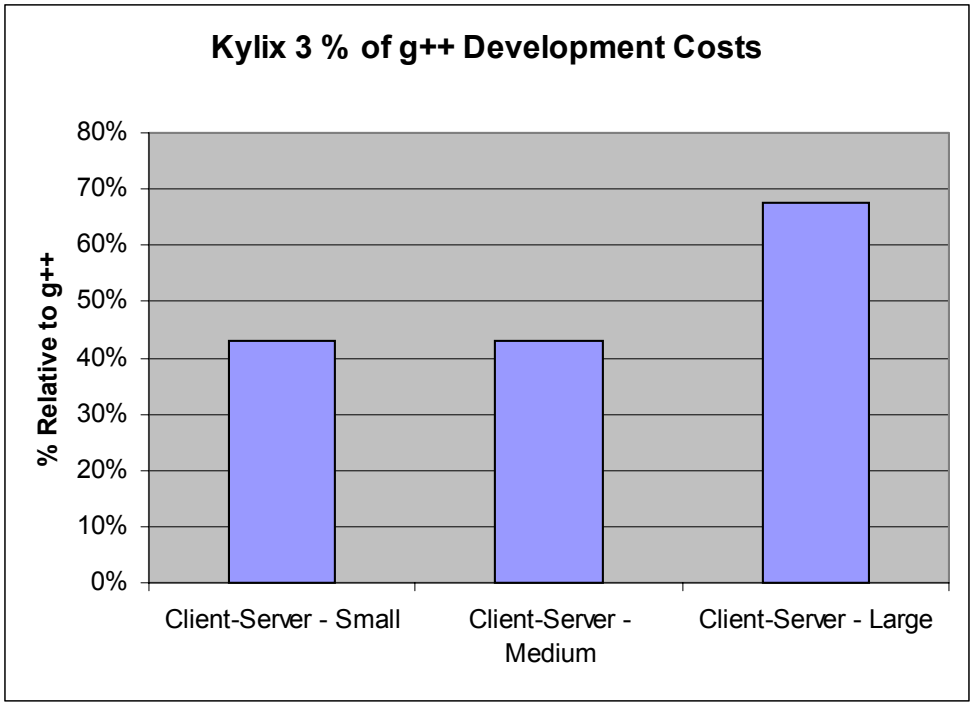
Project	g++	Kylix 3	$ Savings
Embedded	$323,816	$180,685	$143,131
Web Server	$778,286	$314,964	$463,322
Client-Server	$549,055	$235,715	$313,340

The next table shows the results for our analysis of three different sized embedded projects:

Cost comparison–differently sized client-server projects

Project	g++	Kylix 3	$ Savings
Client-Server–Small	$91,840	$39,428	$52,412
Client-Server–Medium	$549,055	$235,715	$313,340
Client-Server–Large	$2,678,923	$1,811,090	$867,833

As shown in the following figure, Kylix 3 saves money on all client-server projects relative to g++ development, with small projects approximately 40% the cost and large projects roughly 68% of the g++ cost.

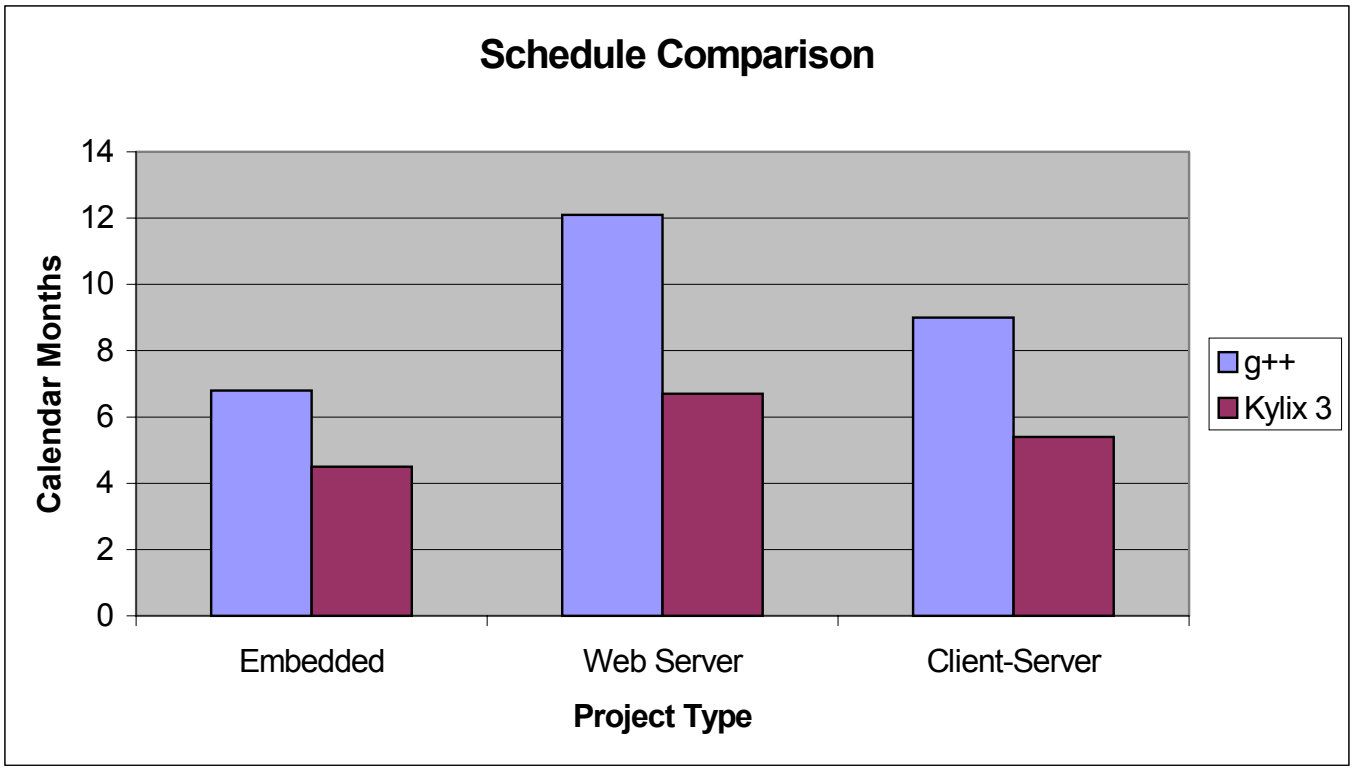


Development time

Because Kylix 3 development requires significantly less effort than equivalent g++ development effort, we would expect that Kylix 3 development projects could deliver the completed application to market faster than equivalent g++. In fact, as shown in the following table and figure, Kylix 3 development is consistently and significantly faster than g++ development.

Schedule comparison

Project	g++	Kylix 3	Time Savings
Embedded	6.8 months	4.5 months	2.3 months
Web Server	12.1 months	6.7 months	5.4 months
Client-Server	9 months	5.4 months	3.6 months



The table below highlights that this relationship extends to all three sizes of embedded development projects studied:

Schedule comparison–differently sized client-server projects

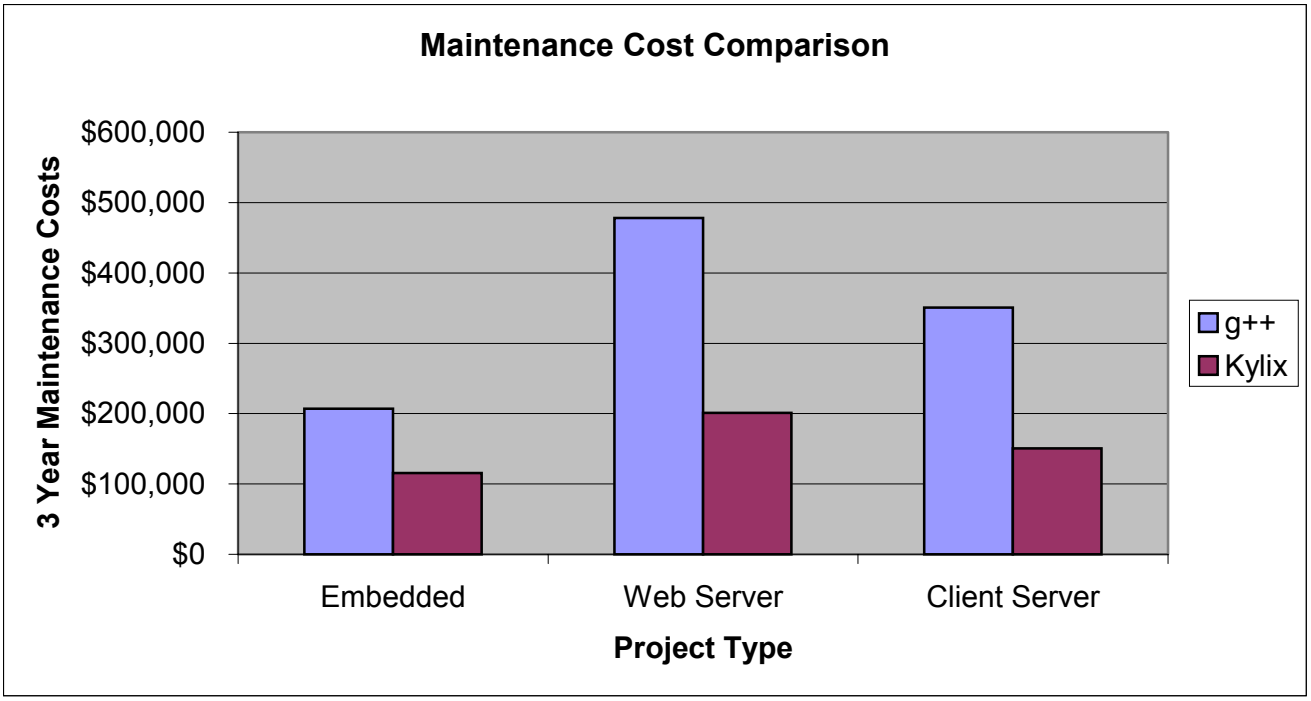
Project	g++	Kylix 3	Time Savings
Client-Server–Small	3 months	2.1 months	0.9 months
Client-Server–Medium	9 months	5.4 months	3.6 months
Client-Server–Large	19.8 months	14.6 months	5.2 months

Maintenance effort following deployment

Of course, development cost only tells a portion of the story. We can appreciate the possibility of saving money during development, but the resulting reduction in software quality will increase maintenance costs downstream. The table and figure below show the projected three-year maintenance costs for our reference projects. As can be clearly seen from the figure, Kylix development not only saves development time and effort, but also reduces on-going software maintenance costs substantially.

Maintenance cost comparison

Project	g++	Kylix 3	$ Savings
Embedded	$206,903	$115,449	$91,454
Web Server	$477,999	$201,246	$276,753
Client-Server	$350,819	$150.610	$200,206

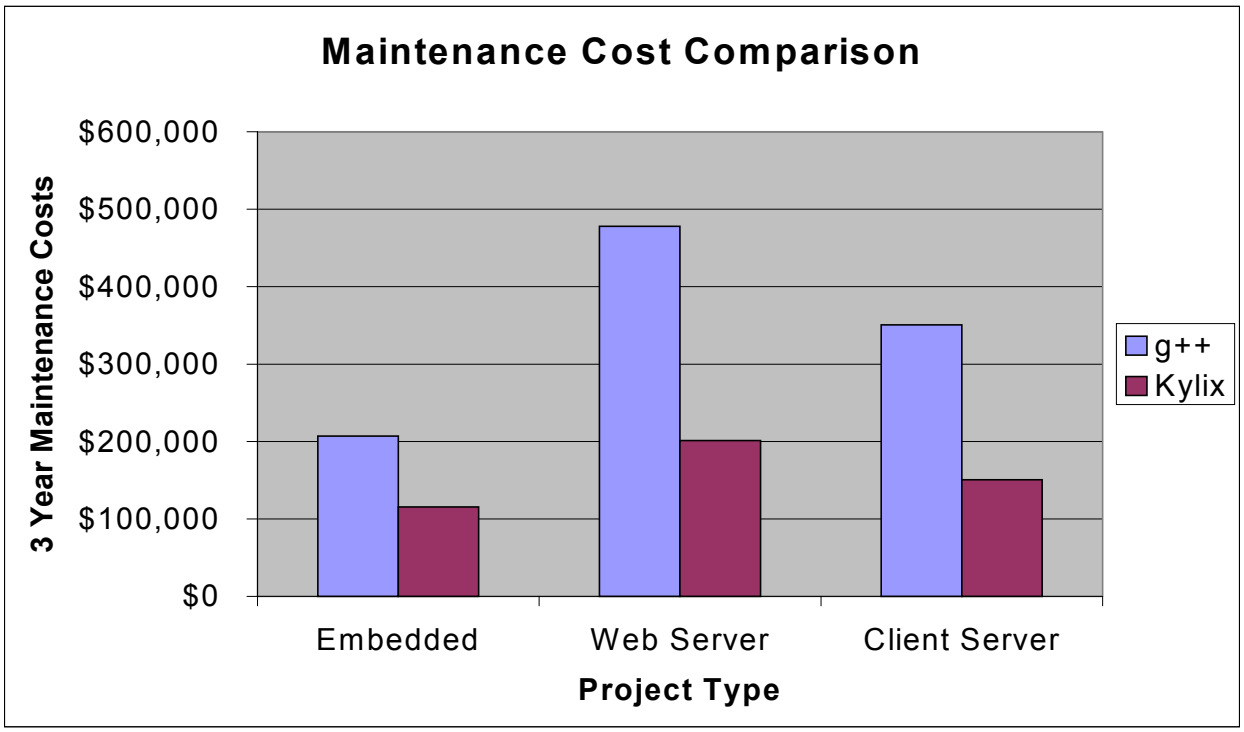


The following table shows that this relationship also applies across all three sizes of client-server development projects.

Maintenance cost comparison–differently sized client-server projects

Project	g++	Kylix 3	$ Savings
Client-Server–Small	$58,860	$25,913	$33,667
Client-Server–Medium	$350,819	$150.610	$200,206
Client-Server–Large	$3,139,764	$1,347,935	$1,791,829

The next figure shows that the reduced maintenance effort is at least partially due to a projected higher quality for Kylix 3 applications relative to g++ applications. The number of residual defects that will be discovered in year one following the models project deployment signifies the higher quality of Kylix 3 applications.



The subsequent table shows the exact number of residual defects that may occur:

Residual defect comparison

Project	g++	Kylix 3	Defect Savings
Embedded	5	3	2
Web Server	102	46	56
Client-Server	160	68	92

Once again, as shown in the table below, this relationship extends to all sizes of client-server development.

Residual defect comparison–differently sized client-server projects

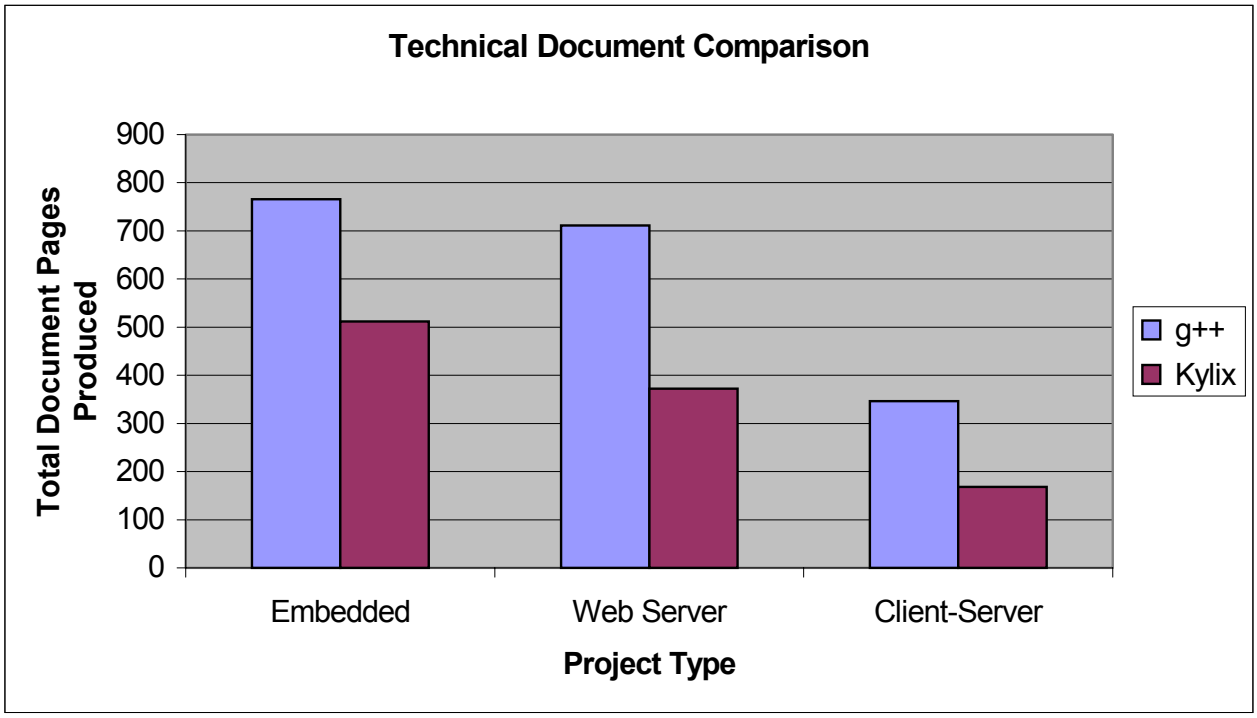
Project	g++	Kylix 3	Defect Savings
Client-Server–Small	26	12	14
Client-Server–Medium	160	68	92
Client-Server–Large	1423	610	813

Technical documentation produced during development

Finally, we looked at the optimum number of pages of technical documentation that the models predict should be produced during development. This documentation is necessary to ensure that the development is successful and to describe the inner details of the software for the maintenance programmers. We would expect that the reduced effort associated with Kylix 3 development, combined with the improved development environment itself, would result in a reduced requirement for technical documentation. In fact, this expectation proved to be accurate as shown on the table and Technical Documentation Comparison graph below.

Technical document comparison

Project	g++	Kylix 3	# Pages Savings
Embedded	766	512	254
Web Server	711	372	339
Client-Server	346	178	168



The documentation needed for specific size client-server projects is as follows:

Technical document comparison–differently sized client-server projects

Project	g++	Kylix 3	# Pages Savings
Client-Server–Small	98	65	33
Client-Server–Medium	346	178	168
Client-Server–Large	2,299	1,084	1,215

Conclusions

There is no question that Kylix 3 offers dramatic improvement in the way software is developed for the Linux platform, across all application domains and for all project sizes that we tested. We observed clear improvements across every metric measured, including effort, cost, time, quality, and life-cycle costs. For the Web and client-server applications, the Kylix 3 effort was about 42% of the effort required for an equivalent deployment using g++. Even in embedded environments, Kylix 3 development could be accomplished in roughly half the effort required for the g++ environment.

This reduction in effort will translate into direct development cost savings. In terms of development time, Kylix 3 achieves better than 40% time-to-market for Web and client-server projects and better than 33% for embedded projects over g++. Finally, when reviewing quality measures and life-cycle cost, we found that Kylix 3 achieves improvements of between 40% and 58% in terms of residual defects and 3-year maintenance costs. These improvements will markedly increase client/user satisfaction and lower the cost of owning/maintaining applications.

Parametric modeling in more detail

Parametric modeling uses roughly 100 input parameters that define a project and more than 70 models of project behavior to forecast the project outcomes. This technique is often used in estimating software project costs and schedules and in preparing optimum project plans. A free sample tool can be downloaded from www.costxpert.com.

Validity of parametric modeling

Parametric modeling of software cost, schedule, and so on has been in existence since the early 1980s. It is employed by most major organizations to help estimate and manage software development projects. The Standish Group, Software Productivity Research, and others have found that the use of these techniques double the probability of projects reaching a successful conclusion. The parametric models used in this study have a design goal of a plus or minus 10% accuracy and are currently achieving an accuracy of (plus or minus) 7% in the field.

Kylix™ 3 productivity is built in from the ground up

Kylix 3 introduces to the Linux operating system a high performance C++ and Delphi language development solution for rapid e-business development. Kylix 3 combines several development tools into a visual development environment that heretofore was primarily the domain of Windows developers. Kylix 3 includes CLX, a component library that simplifies and standardizes reuse of code objects. In total, CLX features more than 190 reusable software building blocks that simplify development greatly. Kylix 3 also includes an advanced code editor, CodeInsight, a fully integrated graphical debugger, and a 32 bit optimizing C/C++ code compiler.

Author Biography

William Roetzheim is the Chief Executive Officer of the Cost Xpert Group, Inc. Cost Xpert Group, Inc. is a leader in the development of parametric software cost estimating tools (www.costxpert.com). Mr. Roetzheim has written 15 books, more than 100 articles, and is internationally recognized as an expert on software cost estimating.

John Amacker is a Senior Cost Consultant with the Cost Xpert Group. Mr. Amacker specializes in independent software cost estimation/analysis and software cost estimation training.

Borland

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000

Made in Borland®. Copyright © 2002 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Linux is a registered trademark of Linus Torvalds. All other marks are the property of their respective owners. Corporate Headquarters: 100 Enterprise Way, Scotts Valley, CA 95066-3249 • 831-431-1000 • www.borland.com • Offices in: Australia, Brazil, Canada, China, Czech Republic, France, Germany, Hong Kong, Hungary, India, Ireland, Italy, Japan, Korea, the Netherlands, New Zealand, Russia, Singapore, Spain, Sweden, Taiwan, the United Kingdom, and the United States. • 13515