Porting Process

What to expect from the migration project with us, how we collaborate with you and how we ensure that the project is successful.

We make the process of porting a Gupta application to Microsoft .NET simple, clearly separated in well defined phases, and easy to manage. With our approach you will be able to assess the progress being made at any time. The outcome of each phase is used as the input for the next.

We do not sell a black box solution, where we convince you to give us your code and wait for us to give you a "turn-key" solution, a large bill, and a lot of work that still needs to be done. We do not complicate things with endless paperwork and excuses to charge more man-hours.

We estimate everything upfront. We define simple and self-contained steps with a clear input and a verifiable output. We guarantee everything we do and if we make a mistake, we fix it at our cost.

These are the major typical phases in a Porting Project:

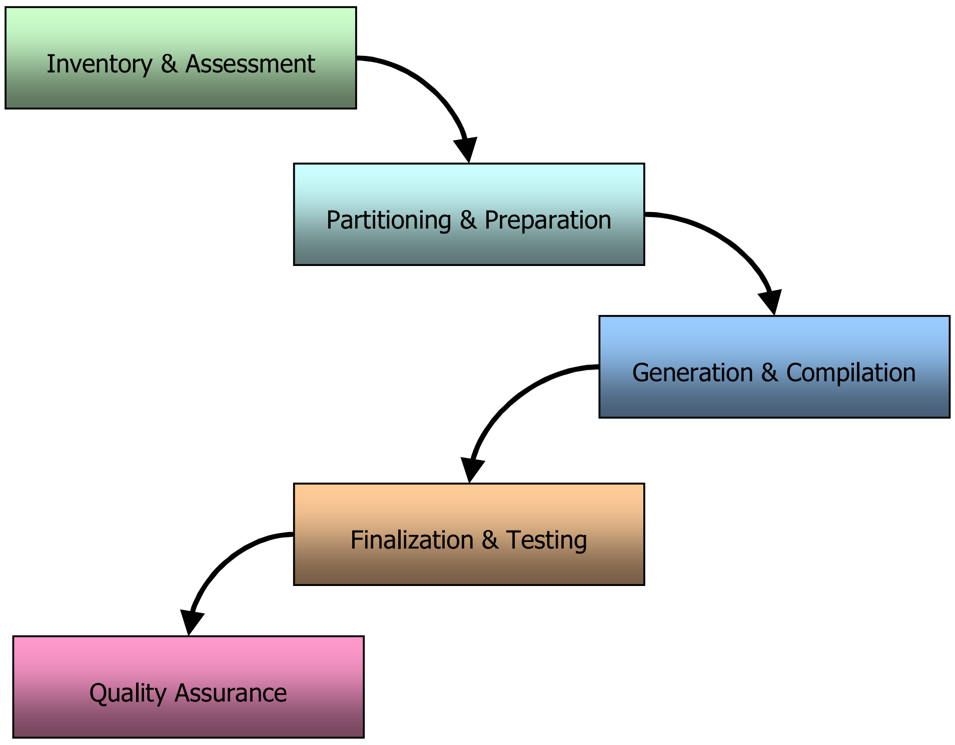


Figure 1 - Phases in a Porting Project

Five Steps to Happiness

We know that it seems simple and that others like to use a lot of arrows, lines, and colorful blocks to describe their super-duper solution. Usually the more fancy the diagrams, the less substance is behind the paper. Our approach, on the contrary, is to Keep It Small and Simple. Or KISS.

We have, in fact, simplified the process as much as possible because our technology and methodology are solid and both have been refined on the field. We don't need to confuse our clients to be able to shift the blame and the unforeseen costs; our process is transparent and we are fully confident in the outcome.

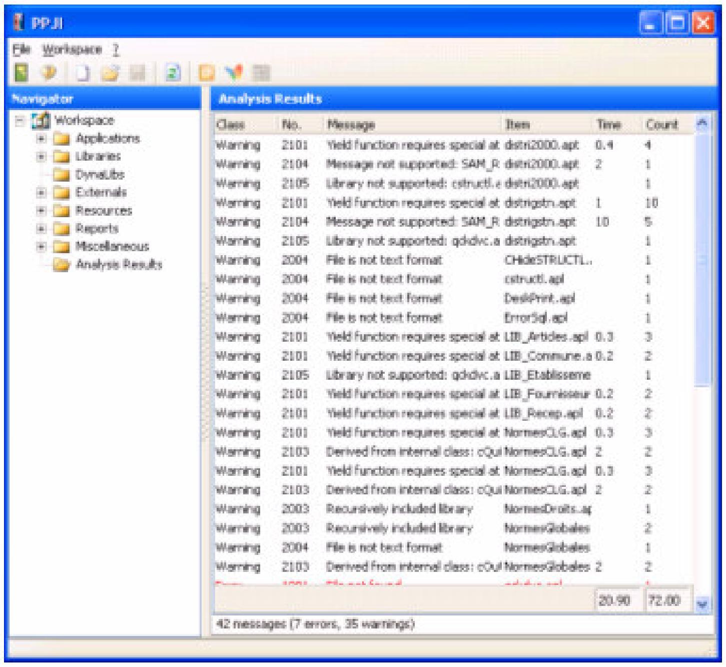
Obviously there are bumps in the road, any software project has problems. The key is to have a technology that is flexible enough to accommodate any adjustment for unforeseen problems, and a process that can detect most problems as early as possible and thus reduce the overall risk to a manageable level.

Here is what we do:

1. Inventory & Assessment (IA)

In the Inventory & Assessment phase, we use our automated tools and professional services (read: experience) to create a full inventory of all the components in the system to transform and to identify all the issues that will require special attention.

The analyzer tool, called PPJ Inventory, generates a list of all known potential problems found in the source code. We look at each issue carefully and evaluate if it can be safely handled by the translation tool and addressed after the conversion to .NET, or if we have to modify the original SAL application in order to increase the quality of the generated code. The end result is to have a higher and better conversion and reduce the overall cost of the project.

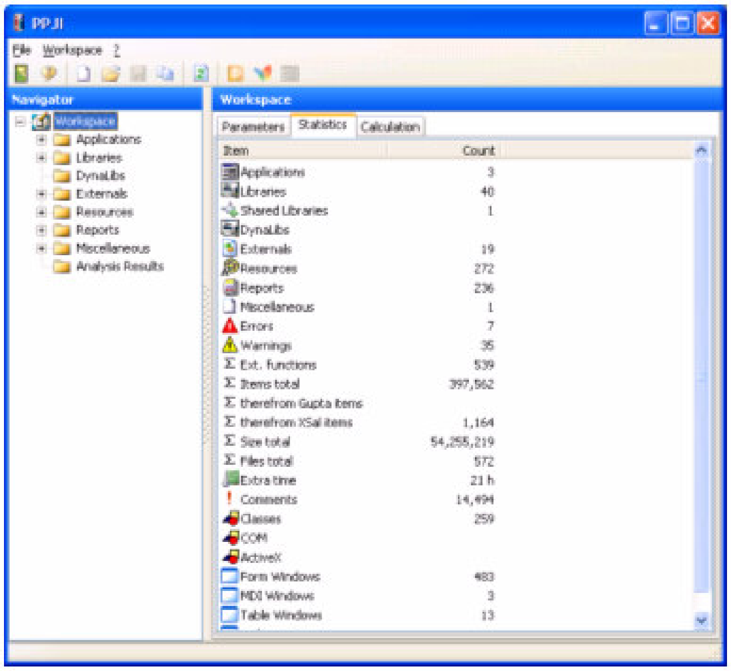


Class	No.	Message	Item	Time	Count
Warning	2101	Yield function requires special at	distri2000.apt	0.4	4
Warning	2104	Message not supported: SAM_R	distri2000.apt	2	1
Warning	2105	Library not supported: cstructl.a	distri2000.apt		1
Warning	2101	Yield function requires special at	distrigstn.apt	1	10
Warning	2104	Message not supported: SAM_R	distrigstn.apt	10	5
Warning	2105	Library not supported: qckdvc.a	distrigstn.apt		1
Warning	2004	File is not text format	CHideSTRUCTL..		1
Warning	2004	File is not text format	cstructl.apl		1
Warning	2004	File is not text format	DeskPrint.apl		1
Warning	2004	File is not text format	ErrorSql.apl		1
Warning	2101	Yield function requires special at	LIB_Articles.apl	0.3	3
Warning	2101	Yield function requires special at	LIB_Commune.a	0.2	2
Warning	2105	Library not supported: qckdvc.a	LIB_Etablisseme		1
Warning	2101	Yield function requires special at	LIB_Fournisseur	0.2	2
Warning	2101	Yield function requires special at	LIB_Recep.apl	0.2	2
Warning	2101	Yield function requires special at	NormesCLG.apl	0.3	3
Warning	2103	Derived from internal class: cQui	NormesCLG.apl	2	2
Warning	2101	Yield function requires special at	NormesCLG.apl	0.3	3
Warning	2103	Derived from internal class: cQui	NormesCLG.apl	2	2
Warning	2003	Recursively included library	NormesDroits.ap		1
Warning	2003	Recursively included library	NormesGlobales		2
Warning	2004	File is not text format	NormesGlobales		1
Warning	2103	Derived from internal class: cOul	NormesGlobales	2	2
				20.90	72.00

42 messages (7 errors, 35 warnings)

Figure 2 - PPJ Inventory with list of detected issues.

Using the PPJ Inventory tool we also generate a comprehensive inventory of all the libraries, dependencies, resources, external modules, and just about anything that we need to know in advance to be able to correctly estimate the work need to complete the project.



Item	Count
Applications	3
Libraries	40
Shared Libraries	1
DynaLibs	
Externals	19
Resources	272
Reports	236
Miscellaneous	1
Errors	7
Warnings	35
Σ Ext. functions	539
Σ Items total	397,562
Σ therefrom Gupta items	
Σ therefrom XSal items	1,164
Σ Size total	54,255,219
Σ Files total	572
Extra time	21 h
Comments	14,494
Classes	259
COM	
ActiveX	
Form Windows	483
MDI Windows	3
Table Windows	13

Figure 3 - PPJ Inventory with inventory results.

During this phase we also translate the entire application to .NET using Ice Porter (the translation tool) and verify all the issues detected by the analyzer. This is a sort of "dry run". It gives us a clear picture of how the translated project will look like and how it will behave in Visual Studio.

The outcome of this phase is a report that gives you a complete view of what's being converted, the organization of the target .NET project, a reliable estimate of the extra work needed to complete the project, and, most importantly, a serious tool for planning and managing the project on your end.

2. Partitioning & Preparation (PP)

Gupta Team Developer applications are usually monolithic programs. While the source code is indeed organized in several files, the finished product is one executable containing everything coming from the source code libraries. Dynalibs never worked very well and are used rarely. Even when dynalibs are used, the source libraries contain forward declarations, circular references, duplicated items, and so on.

In the Partitioning and Preparation phase we organize (separate) the original source code into logical libraries. This sounds more complicated than it really is. We do not modify the source project at all and we do not move the code inside the original libraries. We simply create a parallel set of SAL outlines that group libraries at a higher level creating a modularized "view" of the system. This may be a simple big common library or several layered libraries.

We use the dependencies report generated during the IA phase and the knowledge of your developers to group the shared libraries into complete Team Developer files that correspond exactly to one .NET project each.

The following simple diagram illustrates this process.

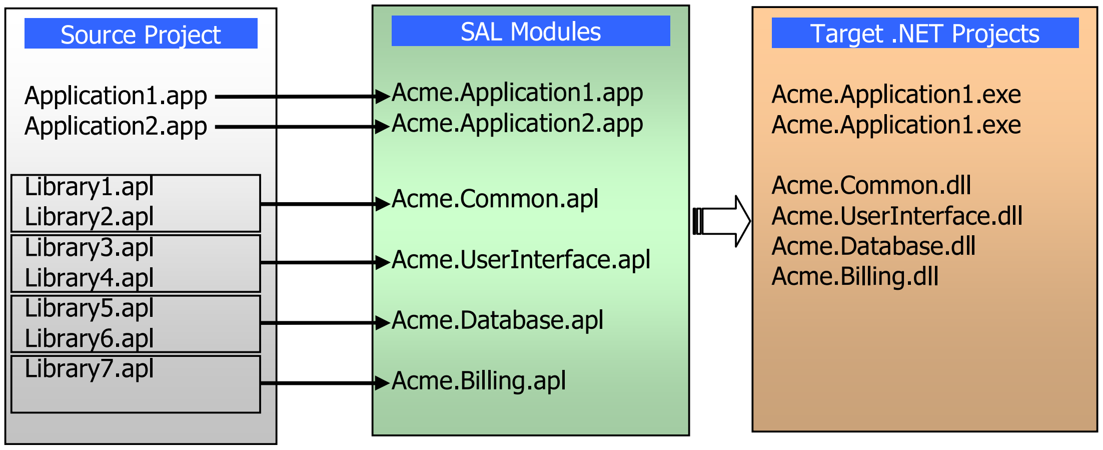


Figure 4 – Sample Partitioning Diagram for an imaginary ACME company.

Clearly the diagram in Figure 4 is simplified. In reality, SAL libraries include other libraries that may, in turn, be included by other libraries. There may be circular dependencies and broken references, as well as duplicated symbols. We are well aware of all the possible complications, and we have most likely already seen it in a project or another.

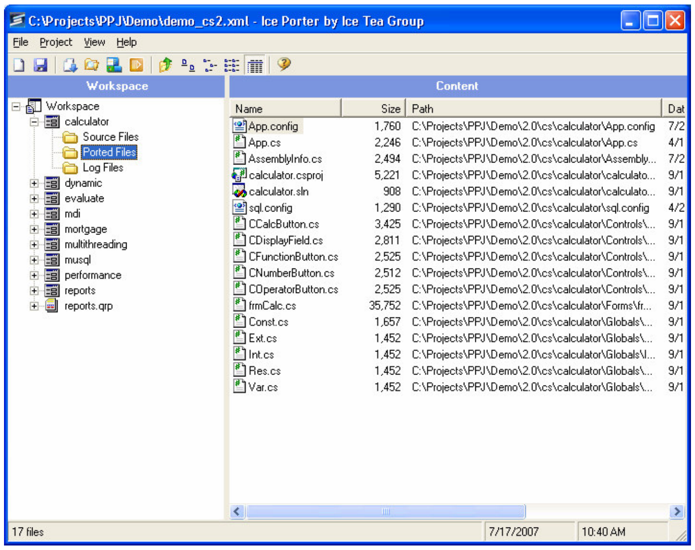
No matter how complex your system is, this technique works and works well. From one single common assembly to several interdependent modules, we have already done it many times.

One of the many advantages of this technique is that the intermediate SAL files can be fully tested and are guaranteed not to alter the original application because they do not contain any code. Additionally, if the original files change during the project, all the intermediate SAL files are automatically updated since the original libraries are simply linked.

The outcome of this phase is a well defined set (could be just one or two, or could be hundreds) of SAL files that can be tested and are ready to be fed into the translation tool in the next phase.

3. Generation & Completion (GC)

Enters Ice Porter™. The amazing translation engine that makes it possible to switch technologies without selling your first-born child.



Name	Size	Path	Dat
App.config	1,760	C:\Projects\PPJ\Demo\2.0\cs\calculator\App.config	7/2
App.cs	2,246	C:\Projects\PPJ\Demo\2.0\cs\calculator\App.cs	4/1
AssemblyInfo.cs	2,494	C:\Projects\PPJ\Demo\2.0\cs\calculator\Assembly...	7/2
calculator.csproj	5,221	C:\Projects\PPJ\Demo\2.0\cs\calculator\calculato...	9/1
calculator.sln	908	C:\Projects\PPJ\Demo\2.0\cs\calculator\calculato...	9/1
sql.config	1,290	C:\Projects\PPJ\Demo\2.0\cs\calculator\sql.config	4/2
CCalcButton.cs	3,425	C:\Projects\PPJ\Demo\2.0\cs\calculator\Controls\...	9/1
CDisplayField.cs	2,811	C:\Projects\PPJ\Demo\2.0\cs\calculator\Controls\...	9/1
CFunctionButton.cs	2,525	C:\Projects\PPJ\Demo\2.0\cs\calculator\Controls\...	9/1
CNumberButton.cs	2,512	C:\Projects\PPJ\Demo\2.0\cs\calculator\Controls\...	9/1
COperatorButton.cs	2,525	C:\Projects\PPJ\Demo\2.0\cs\calculator\Controls\...	9/1
frmCalc.cs	35,752	C:\Projects\PPJ\Demo\2.0\cs\calculator\Forms\fr...	9/1
Const.cs	1,657	C:\Projects\PPJ\Demo\2.0\cs\calculator\Globals\...	9/1
Ext.cs	1,452	C:\Projects\PPJ\Demo\2.0\cs\calculator\Globals\...	9/1
Int.cs	1,452	C:\Projects\PPJ\Demo\2.0\cs\calculator\Globals\I...	9/1
Res.cs	1,452	C:\Projects\PPJ\Demo\2.0\cs\calculator\Globals\...	9/1
Var.cs	1,452	C:\Projects\PPJ\Demo\2.0\cs\calculator\Globals\...	9/1

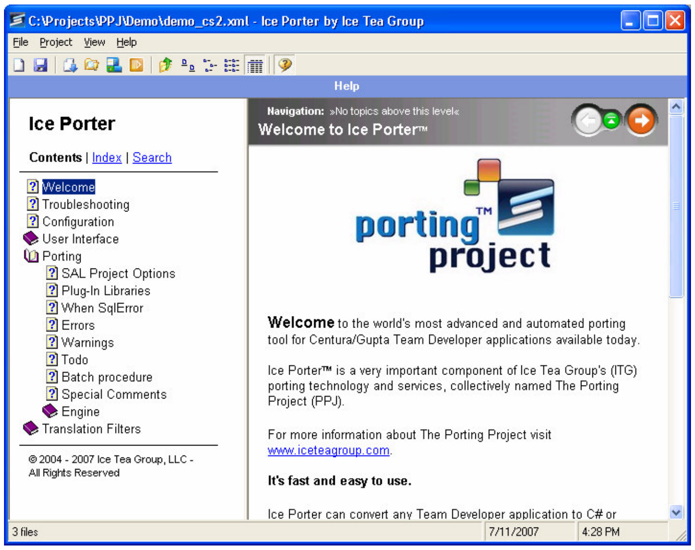
17 files | 7/17/2007 | 10:40 AM

Figure 5 - Ice Porter in action.

The outcome from the previous phase (Gupta files ready to be translated) is used to configure the project space in Ice Porter. Each SAL intermediate file corresponds to one .NET project in Visual Studio. Each project must compile without errors.

Starting from the bottom-up, all the projects are ported and compiled. Ice Porter takes care of incrementally removing all the SAL code that has already been ported in shared modules. The result is a set of Visual Studio projects with virtually no duplicated code, clear and fully qualified references among the modules, and a new clean and well organized project tree. All the projects can be loaded at once in a single Visual Studio solutions and further code reorganization can be accomplished by simply dragging & dropping classes from one project to another.

Our migration experts make sure that the code generated by Ice Porter is fully compilable and runnable. Every trouble spot is isolated and clearly tagged for further processing. We have a worldwide network of partners and migration experts trained on Gupta Team Developer, Microsoft .NET and our porting technology.



© 2004 - 2007 Ice Tea Group, LLC - All Rights Reserved

Welcome to Ice Porter™

Welcome to the world's most advanced and automated porting tool for Centura/Gupta Team Developer applications available today.

Ice Porter™ is a very important component of Ice Tea Group's (ITG) porting technology and services, collectively named The Porting Project (PPJ).

For more information about The Porting Project visit www.iceteagroup.com.

It's fast and easy to use.

Ice Porter can convert any Team Developer application to C# or

Figure 6 - Ice Porter showing the built-in help.

Ice Porter is an advanced and mature tool that can port any Gupta application to C# or VB.NET. The tool also supports plug-in translation filters for customized transformations. The filters can be written in any .NET language and can manipulate the source Gupta code through a .NET CDK interface, the intermediate CodeDom structures (XML-like code tree) generated by the tool, the output text files, and Visual Studio project using the DTE interface.

The outcome of this phase is a complete Visual Studio .NET solution that fully compiles. This will be input of the next phase, which takes care of smoothing things out.

4. Finalization & Testing (FT)

The old 80-20 rule about software development says that 80% of the time and effort is spent on last 20% of the project. While the 80-20 proportion is a metaphor, the general concept is correct also for porting projects.

Finalization and Testing will most likely account for a large chunk of the overall time and effort of the entire project. With most of the effort spent on small details. This is also why we stress over and over that details are important and you cannot simply write off a particular Gupta feature simply because there may be something vaguely similar in the plain vanilla .NET Framework.

In any case, if we are at this stage it means that now we have .NET project(s) that compile and basically run. We don't have to work on Gupta code anymore. We may need the original Gupta application for testing and comparing code flow, but from now on all our coding will be in .NET.

As the project manager for a very important company said about his developers "*They have already forgotten about Centura Team Developer. They look happy!!!!*"

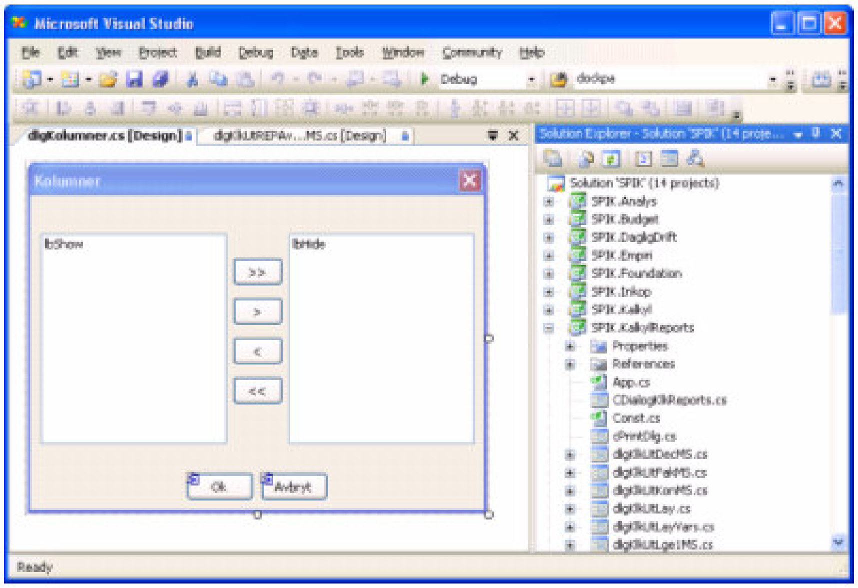


Figure 7 - Several projects in a Visual Studio solution.

Finalization work is very important and must be carried out sistematically. Our method is to split the work in logical groups and thoroughly work on each group. For example: all forms and dialogs have to opened successfully in Visual Studio designer and all labels must be verified; all menus and menu actions must be exercised; all missing functionality must be implemented; all button actions, combobox, listboxes, and validation must be exercised, etc.

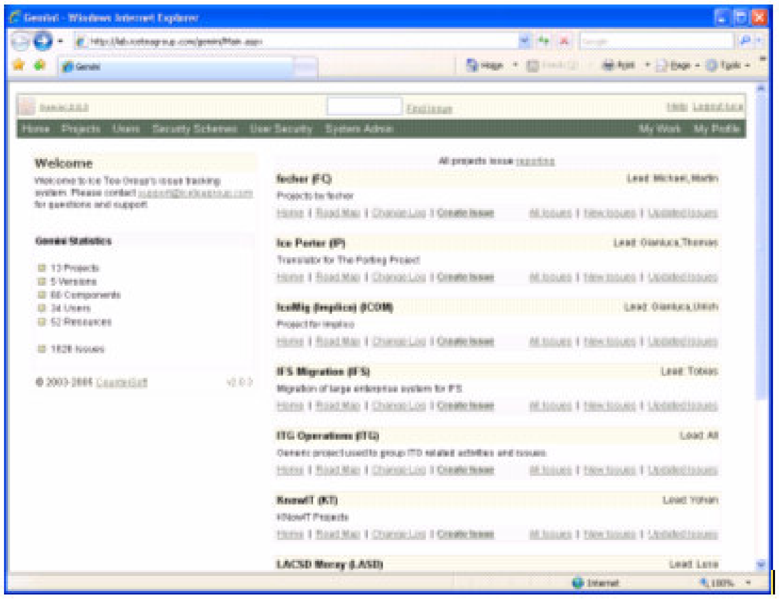
At this point we can also use one of the many unit testing tools that are available for .NET and create unit tests for the ported code. Our favourite tool is NUnit, but there are many others that do a superb job. Another great advantage of switching technology.

The output of this phase is an application that is ready for being deployed to the QA people. Typically so-called power users that will do their best to break the application.

5. Quality Assurance (QA)

The Quality Assurance phase, a sort of beta release, is the last step before full deployment. This phase is carried on by you (the customer) with our support to debug and fixed the defects that are reported by the power users.

We use an online issue tracking system that is opened to our clients and partners to keep track and address every single issue. The issue database is also used to log defects related to the PPJ Framework and make sure that whatever can be fixed at the lowest possible level is fixed there instead of having to run after the same defect manifesting itself in different areas of the application.



Welcome

Welcome to Ice Tea Group's issue tracking system. Please contact [illegible] for questions and support.

Gemini Statistics

- 13 Projects
- 5 Versions
- 66 Components
- 34 Users
- 52 Resources
- 1828 Issues

© 2003-2006 CounterSoft

All projects issue [illegible]

fecher (FC) Lead: Michael, Martin
Projects for fecher

Ice Porter (IP) Lead: Gianluca,Thomas
Translator for The Porting Project

IceMig (Implico) (ICOM) Lead: Gianluca,Ulrich
Project for Implico

IFS Migration (IFS) Lead: Tobias
Migration of large enterprise system for IFS

ITG Operations (ITG) Lead: All
Generic project used to group ITG related activities and issues

KnowIT (KT) Lead: Yohan
KNowIT Projects

LACSD Money (LASD) Lead: Luis

Figure 8 - PPJ Issue tracking system.

After step 5 (QA) your application will be a Microsoft .NET application with all the inherent pros and cons. We see this as the beginning of a new life for your application, not merely the end of the porting project.

After the dust has settled and the application is fully running in .NET, you have a huge world of IT options to choose from. Basically your application will say, once again, **Hello World!**

Working together

Our porting methodology is flexible and it's adapted to the unique needs of each customer and each project. The goal is to complete the best possible conversion at the lowest possible cost. Our solution costs less than any alternative not because we do it on the "cheap", but because we focus on the technology, automation and collaboration, which allows us to distribute the cost among all projects and reduce the overall risk for all projects.

We know very well that you have a unique knowledge about your systems and we certainly want to reuse it. Most of our conversion experts are developers and consultants that have also worked for (or are still consulting for) companies with Gupta applications. We know first-hand what it means to have an application that has been developed over many years and that contains very valuable business knowledge. The code in the original application is the product of countless hours of programming and testing. To throw it all away and restart from beta 0.1 doesn't make sense for us.

Our process is setup to let us carry on the phases where we can be more effective, and to use your knowledge and developers where you can be more effective. The goal is to reduce the external costs and maximize existing knowledge.

This is how we achieve our goal:

> During the Inventory and Assessment phase we interact with your IT people to make sure that we have an accurate picture of the system. One of the outcomes of this phase maybe a list of simple changes that can be applied to the source applications and that will reduce the overall cost of the project.
>
> During the Partitioning and Preparation phase we interact with your IT people to either gather enough information to achieve a better separation of the system into modules; or we simply support your IT department and let you organize the code to be ported following our general guidelines.
>
> During the Code Generation and Compilation phase we may retranslate the same code several times and collaborate with your developers and managers until we have generated the best possible code that meets your specifications.
>
> The Finalization and Testing phase is where we collaborate with the customer the most. In many cases this phase is entirely carried on by the customer with our support.
>
> In the Quality Assurance phase we are only on the receiving end of the project. Our job is to quickly address any issue that you report and to make sure that each issue is investigated well and deep enough to reduce QA time and avoid having to address the same problems in different areas of the application.
>
> After the project is completed, we keep maintaining and updating the PPJ Framework (our compatibility library). We make sure that all known defects are addressed and that the library is up to date with all new .NET releases, ADO.NET drivers, and Windows versions.

Conclusions

Changing the underlying technology of an application is not easy and it carries many risks. We believe that we have the most advanced technology as well as the most flexible methodology to achieve a virtually risk-free migration.

Most importantly, we give you all the tools, all the information, and all the checkpoints that allow you to make the right decision and to monitor the project very closely.