# WHAT'S SO GREAT ABOUT DESIGNER?

*Marc de Oliveira, PYTHIA Information*

This paper will explain the reasons why Designer is such a unique product that is not matched by any other development tool. The reason for doing this is mainly to let new Oracle developers know why some of the experienced Oracle developers keep talking about how fantastic this product is, while Oracle is almost ignoring it completely.

Another reason is a small hope that some key product managers of Oracle might realize what they have in Designer and maybe not write it off as a product of the past that only suites outdated technologies and development methods.

Because Oracle is hoping to implement the important features of Designer in future versions of JDeveloper I will also be relating some of the key features of Designer to the architecture of JDeveloper.

## The history of Designer - in brief

I began using Oracle*CASE (that was the name of Designer at the time) in 1990 when it was at version 4.2. The user interface was character based and the Forms Generator generated Forms 3.2. Besides that everything was pretty much as it is today.

A couple of years later, in 1993, I saw Dai Clegg make a presentation claiming that Oracle*CASE would become an Object Oriented development tool. His point was that the Entity would evolve into a Class that would become "an encapsulated definition of information and operations". Sub typing would evolve into Inheritance and Oracle would support objects in the database.

In 1995 Designer 1.0 was released. Beside the Forms and Reports generators it had an MS Help file generator. The main difference, though, was the new graphical user interface that Oracle hoped would make Designer an easier product to use. At the same time Dai was talking about Sedona - the project that would merge Designer and Developer into a single repository so that the Forms generator could become even more effective by allowing the developer to control the generated form programmatically.

In 1996 Sedona seemed to be forgotten and Dai Clegg was talking about Valhalla. Valhalla was the code name of an Oracle version of Borland's JBuilder for building Java programs that ended up being released as JDeveloper. He seemed very impressed that JBuilder did not have an application model but that the code *was* the model and vise versa. Even today Oracle employees still use every chance they get to say "When you change the model the code changes, and when you change the code the model changes".

Designer 2 came out in 1997. It had a Visual Basic generator, a C++ generator (that was removed in the next version of Designer) but the most interesting new generator was the Web Server Generator that generated HTML and JavaScript for building light weight web applications.

The next big leap for Designer was Ian Fisher's SCM product, introduced with Designer 6i in 2000, that allowed us to keep track of all versions of every Repository element.

After that Oracle's interest in Designer has been dropping the reason being that Designer is now a "mature" product that does not need much development other than a little bug fixing. Instead Oracle's current claimed focus is to implement the model based features of Designer in JDeveloper. In time JDeveloper should replace Designer.

## So, what *is* so great about Designer?

How can it be that some developers keep using a product that seems abandoned by its maker? Shouldn't there be a newer product somewhere that could replace it?

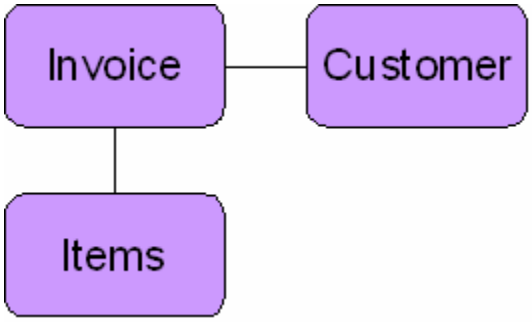
Well, I don't think so. I believe that Designer is still the only product that can fully handle all phases of the development cycle in a completely integrated way.

You have better upper case products than Designer and you might even have better code generators than Designer's. You certainly have more advanced project management tools than Designer and there might be better development methods than the ones supported by Designer. But still, Designer is the only product that has a complete architecture that covers all these parts, so that any element defined throughout the development process is only located in one place and that changing any element will automatically be reflected in all related parts, like documentation, models and code.

Taking the extensibility features and API of Designer into account, you can take Designer anywhere, you need it to go. A number of third party tools are available that add functionality and whatever is not there you can build yourself or get someone to build for you.

In the following sections I will elaborate a little on what I find to be the ten most important features of Designer.

## Model Driven Development

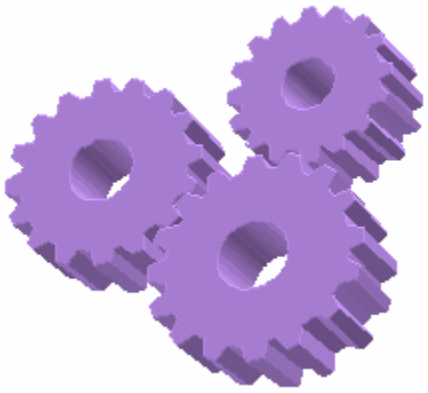



The key feature of Designer is obviously its support for model driven development, and this is obviously a feature that Oracle is very interested in implementing in JDeveloper as well.

Model driven development is not just the ability to draw pretty diagrams. To me, it requires the model to be detached from the code because a very important benefit of model driven development is the independence of the implementation platform. When you can model your business requirements on an abstract level that model could be implemented in numerous ways. Even in ways that are not invented at the time of designing the model. An application model describes what the application needs to do but leaves it to others to actually implement the application on a specific platform.

To me it seems impossible to ever achieve this in a tool like JDeveloper where the model and the code is so tightly integrated that they are actually the same. Imagine if Java and OO are not the last development paradigms we ever need to learn! I know, it seems hard to believe, but it has happened before ☺…

## Code Generation



A good application modeling tool and a good set of generators makes up a killer combination. And that is almost what we have in Designer.

Rather than having the model being the code, we want to have a whole set of generators that can implement the model on different platforms. Most importantly the possibility to build new generators at some later time assures that the applications do not get stuck in some old "technology swamp" where it can no longer be maintained. Just plug in a new generator and generate your application into whatever platform necessary. Of course, this regeneration thing is not free – you might have to solve issues about incompatibilities between the platforms – but it sure beats having to recode everything from scratch.

### Home Grown Generators

If the generators shipped with Designer (Server, Forms, Reports, Web Server, Visual Basic and MS Help) are not enough you can always build your own generators or use generators built by somebody else.

HTML Help, MS Word, ASP and PHP generators are only a few of the generators already made by individual developers.

It is not difficult to build home grown generators because Designer comes with a very well documented data model of its architecture and a simple API for accessing all elements of the repository. This possibility is made even easier with the Common Designer Generator Interface that facilitates both the building and the sharing of home grown generators (see my paper on the Common Designer Generator Interface elsewhere in the ODTUG 2004 proceedings).

### Standards

Designer provides a framework of preferences that can be organized in preference sets to let you control the behavior of the generators. Together with the ability to use domains, libraries, templates and style sheets you get the possibility to have the generators conform to the individual user interface standards of your organization. This is a very important feature to assure that standard functionality and look-and-feel is not applied on each module by individual developers.

Rather than use generators to get a "first cut" program that can then be altered to meet ones needs, all specific design and coding needs should be implemented through the above control elements so that the correct code is created directly by the generators. This strategy will assure the maximum level of reuse and a consistent look and feel throughout ones application systems.

It will also make it much easier to implement changes in your coding standards or user interfaces, and get the maximum benefit out of improvements to the generators, because you will be able to regenerate the code at any time without having to re-implement alterations.

JDeveloper's approach is to have wizards that help developers make the "first cut" program. This approach makes it much harder to re-implement modules to conform to a new standard or platform, as the altered code could be difficult to identify and re-implement.

To get the best chance of generating your applications directly out of Designer without the need of changing the generated code you should use a professional template and library package like DesignAssist or Headstart.

## Open API


Designer is shipped with both a pl/sql API and a command line API giving access to retrieve and update information about all 400 element types of the Designer repository. Basically, if you don't like the Designer user interface you could build a completely new interface in the language of your choice.

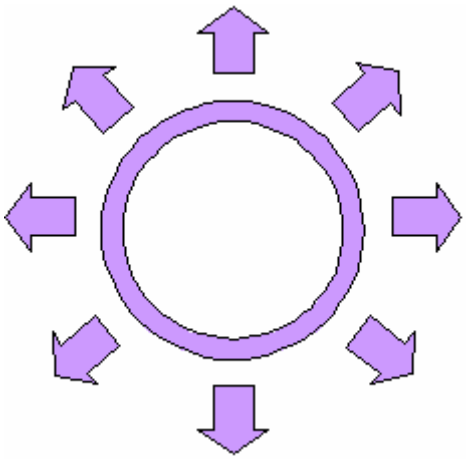
Using the Designer API it is possible to build your own quality assurance software that controls if the naming and coding standards defined by your organization are met. You can even enforce coding standards or implement programmatical alterations or additions to the repository.

A couple of years ago I made a program that took a Designer Reports module and created a corresponding Forms module in Designer that would generate into a parameter form for calling the Reports module (see my paper "Make Designer 2.1.2 Generate Your Parameter Form" from the ODTUG 1999 conference).

So, whatever you need Designer to do that it is not already doing might easily be done using the API.

This is another part that JDeveloper will have big difficulties in implementing because JDeveloper does not have a repository. Everything is defined in XML files over which you don't get the same kind of control.

## Extendable



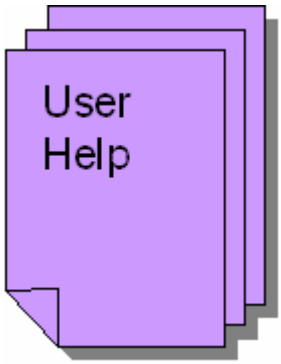
The API will let you add functionality to Designer but you can also extend its data model.

The User Extensions of Designer allow you to add up to 20 new properties to each of the 400 types of repository elements. For example you could add title, first_name and last_name properties to the Schema element if you chose to register your developers as schemas.

The User Extensions also let you add new relationships between all element types. This feature would let you define the assignments of modules to each developer.

Whenever considering extending the data model you often have the choice of building new tables outside the Designer repository instead. The best choice will always depend on the individual situations, so be sure to think your situation through carefully before making a decision.

## Documentation




Because of the User Extensions and the file management features of Designer you have the possibility of storing all documentation related to your development projects in the Designer Repository. You can even define relationships between the external document files and the Designer elements they are related to.

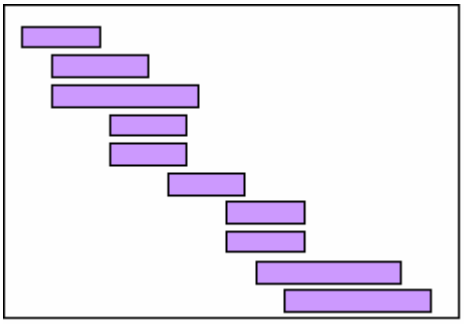
Doing this will give you a perfect overview of the complete set of documentation related to any part of your application systems, and you will easily be able to identify the parts that will need changing as a consequence of modifying your requirements or your application.

An even better solution is to actually generate all the documentation out of the repository. If you can manage to store all data related to your development project in the repository then it should be possible to extract the relevant information on the fly. So your requirements document should basically be a list of all the requirements stored in the repository and your user's reference guide should be a structured collection of all the help text created for each model etc. By building your documentation directly from the repository data you will get complete and up to date documentation of your development project, and every time something changes you will be able to just regenerate all your documentation. This method will ensure that everybody in the project is always on the same page and that nobody are working on outdated information.

Another benefit of building your documentation out of the repository data is that you could build the documentation in multiple formats for printing, for viewing on the net, for working on a spread sheet etc.

To do this you could either simply use the repository reports that are shipped with Designer or you could build your own if you want more control of the layout of your documentation. CaseTech has built an advanced framework for building detailed documentation in HTML out of the repository, and I built a generator for creating documentation out of the repository in MS Word format (see my paper "Generating Repository Reports in MS Word" from ODTUG 1999).

## Project Management features

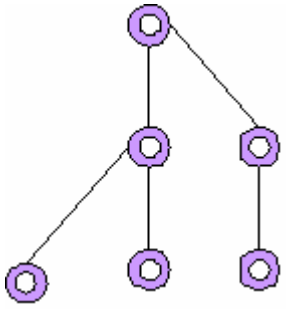


The built-in project management features of Designer are not the most advanced to say the least. Yet, the fact that the project management information is stored together with the elements being developed makes the construction very interesting. We all know about the grief of maintaining Gantt diagrams in MS Project to match the actual progress of the development project. By using the module properties like Status, Complexity, Size, Activity, Task and Time Estimate you can extract data about the development progress that the individual developers maintain themselves. And with a little creativity you can even generate an MS Project Gantt diagram out of the repository (those can be exported and imported in simple text format, you know).

I added a number of project management features like worksheets for developers, progress overviews, deadline overviews etc to Designer in this way (see my paper "Project Management with Designer – How To Do Everything with Designer" from ODTUG 2003).

Even though the built-in project management features of Designer are not very good it is possible to extend them to match your needs, while JDeveloper do not have anything resembling project management capabilities, and without a repository in JDeveloper I don't see how it will ever be possible to implement any project management functionality.

## Versioning



Having all the development project information stored in a single repository requires a pretty advanced configuration management strategy. Again Designer rises to the challenge by offering a very detailed versioning, configuration and security package called SCM.

SCM allows you to version all primary elements like tables, views, modules etc (ie not lower level elements like columns, items and arguments).

It also offers the notion of Workareas that makes it easy for developers to manipulate the elements they need to work on. So if changes are required to elements as they looked on a specific date you can setup a Workarea that shows all elements as they looked at that time. You can check those elements out, change them and merge your changes into the current version of

the same elements. Very advanced rules for Workareas are possible, so just make sure that you know what you are doing! One thing that cannot be done, though, is to include more than one version of an element in a single Workarea (it would not make sense, anyway).

SCM also supports branching, so that you can apply multiple changes to the same element in parallel, and then merge them into your main branch as the changes are completed and approved.

Another important SCM term is the configuration. Configurations allow you to stripe all the elements that make out a release and store all the version numbers of every included element, so that you can easily add patches to any given release by creating a Workarea on its configuration.

An important thing missing from the versioning feature of Designer is the lack of traceability into the generated elements. It would be very desirable if for example a generated form somehow could show the version number of the underlying module as it was during the generation.

## Supports a Methodology


Case*Method was a methodology developed by Richard Barker and Dai Clegg at the time Oracle*CASE was built. Case*Method is supported by Oracle*CASE (and Designer) but it is not specific to Oracle*CASE or Designer. The method could be implemented using other tools and other methods can also be supported by Designer.

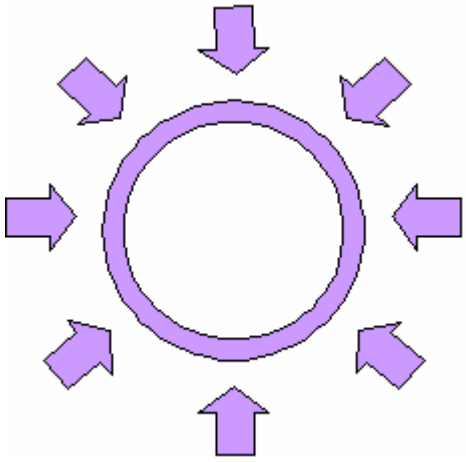
Some of the best development methods supported by Designer are the CASE Application Development Method (CADM) created by Peter Koletzke and Dr. Paul Dorsey in their book Designer Handbook (there is also a RAD version of the method), the CDM Rule Frame by Oracle Consulting in the Netherlands and the CaseTech 4-Step RAD that includes a clever business rule technique for requirements gathering and coding.

As long as you comply with the basic architecture of Designer you are free to define your own method or maybe just refine one of the existing methods to suite your requirements.

Support for the management of bug reports and enhancement requests related to applications developed using Designer is not offered by Designer, but with User Extensions and the Designer API this kind of functionality can also be integrated into the tool. You are only limited by your imagination.

JDeveloper is trying to support Use Case as a methodology but currently this methodology only includes a Use Case diagram with underlying text attributes for documentation. To my knowledge, the Use Case models in JDeveloper are not translated into actual code. They are strictly used for documentation.

## Full Integration



As all the elements of Designer are residing in a single centralized repository, every part of the product will have equal access to the same data. No integration is necessary between the different parts as they all share the same information.

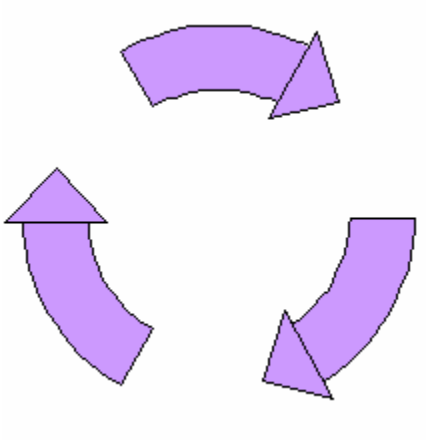
The fact that any change made by one part of the product may influence all other parts obviously adds to the complexity of working with a tool like Designer but in my view this high level of integration is the key to the huge benefits you get from using Designer – provided that you use it correctly ☺…

By the way, I think that an important reason that Designer never became the popular product that Oracle wanted it to be is related to this. When Oracle*CASE became Designer, Oracle focused a lot on the new graphical user interface that should have more appeal to customers than the old character based screens. Obviously, the graphical interface was better than the old one but a misconception made by Oracle sales people was that Designer therefore had become a simple tool to use.

By pushing this misconception unto the customers, Designer was conceived to be a tool for unskilled developers to generate screens and reports by just drawing some simple diagrams. Quite like Oracle later pushed WebDB, and like they are currently pushing HTML DB. These are products to build simple applications by just pointing and clicking a couple of times. Designer was never anything like that.

Developers that knew Forms and Reports or managers that needed to build complex applications did not perceive Designer as a tool for them, while the companies that brought in Designer to help their less skilled developers quickly came to the conclusion that the product was way to complex for them to learn and they ended up just using it for making data diagrams.

### Full Life Cycle



Adding up all the mentioned benefits of Designer, you get a tool with true support for the full life cycle of the development process from the initial vision specification through analysis, design, implementation, documentation, installation and maintenance including bug fixing and patching.

I don't think that any other tool combines the support of such a wide range of development related tasks with the complete integration of all functionality given by having a single centralized repository.

## Conclusion

I hope to have given you an understanding of the uniqueness of Oracle's Designer and why I am reluctant to believe that JDeveloper as it currently looks, will replace it.

This does not mean that I am against JDeveloper replacing Designer. I have no preference as to the name of the tool, so if JDeveloper becomes a true full life cycle development tool I will be happy to use it instead of Designer, but I cannot see this happening without serious changes in the JDeveloper architecture. Firstly, it is absolutely vital that JDeveloper gets a repository for storing metadata. Pretty diagrams stored in XML will not cut it. Secondly, the code needs to be separated from the model so that the product becomes more open to different implementations of the models. Thirdly, the wizards need to be replaced by real generators to avoid manual code being scattered all over the place.

A signal from Oracle confirming that these kinds of changes are being considered would surely be a big relief to many Designer users.

But even if these important aspects of Designer eventually do make it into the JDeveloper toolset it will take years for it to happen, so for the time being I will be sticking with the only truly integrated full life cycle development tool: Oracle Designer.