



N.CYCLES

software solutions

AppOnTap White Paper

Accelerating Development Velocity with AppOnTap
Code Generation

September 2000

65 Germantown Court
Suite 205
Cordova, TN 38125
Phone 901-756-2705
Fax 901-758-1597

1616 West Gate Circle
Nashville, TN 37027
Phone 615-403-3168
Fax 630-214-4909

www.ncycles.com

Table of Contents

Table of Contents	1
Introduction	2
“From Scratch” Limitations.....	2
Typically applications are built in the following five phases	2
This is just the way things are	2
Code Generation via AppOnTap.....	4
How Does APP on TAP Work	5
What’s the Real Difference?.....	6
Advantages of App on Tap	7

Introduction

Does this describe your last IT project? Late, over-budget, incomplete, incorrect, full of bugs, the list can go on and on. In today's Information Technology departments there are many different ways that new computing systems get built. The results of these efforts vary but mainly consist of a series of failures or false starts, expending large amounts of money on both internal and external resources; this paper will describe a new way of implementing computing systems.

“From Scratch” Limitations

Typically applications are built in the following five phases

- **Analysis:** This phase is typically where the business requirements are gathered, the data model is determined, and the high level functional breakdown is created. A first cut at business rules is created. This phase is typically done at a high level with subject matter experts (SME) in the business area being analyzed. These experts typically do not have experience or knowledge of developing application systems.
- **Design:** This phase is where the functional breakdown is finalized and data elements are assigned to different screens. The server model is created and validated, and business rules are finalized. This phase is done using the results of the analysis phase and is typically performed independent of the SME, however it is validated by the SME. Functional specifications, data flows and architectural diagrams are the standard deliverables.
- **Construction:** During this phase the system is coded and unit tested. The programmers perform this testing to ensure the individual units are working as the design specifications describe.
- **Testing:** The testing phase is where all the units of the system are put together and tested together. Typically the SME's are involved again to ensure that the system performs as desired. The programmers are not directly involved in this testing but are available to make corrections as necessary.
- **Implementation:** The final phase in any project is actually implementing the new system. There is typically an implementation plan and schedule as well as any final data conversion performed that may be necessary. The lead analysts or programmers typically perform this.

This is just the way things are

There is nothing really wrong with this model or approach to application development. There are however, several items that need to be considered. First, many IT departments are outsourcing application development. This brings several other factors into play, most notably the cost factor becomes extremely important in the total process. Incorrect or incomplete specifications or analysis can become extremely costly in re-writes or changes to code during later steps in the process. Granted this can happen when in-house

resources are being used, but these resources typically do not cost as much as consultants. We will examine the pricing of projects using this methodology later in this section.

A second drawback to this methodology is the total time involved in following it, and more importantly the time involved in re-doing phases that were done incorrectly. The key to following these steps is to get them right the first time, because changes to requirements or specifications after coding or testing has been performed can lead to double or even triple the amount of time expended to correct the errors. This methodology requires the analysts performing the analysis and design to spend extra time and be extremely detail oriented and careful when performing these steps. This does not usually occur because these two phases of the project show the least amount of tangible progress. There are no screens to look at, there is no amount of code to point to and therefore these two phases of the project tend to get done quickly and poorly to get to the phase that does show progress. It has also been our experience that the majority of people actually performing the analysis and design do not do it correctly or do not have the experience to perform these steps right.

Now, let's examine how a project of this nature typically gets priced. There are really two possible scenarios, each with their own drawbacks. For purposes of these examples let's assume all the resources that will be on the project will bill at the same rate and that they cost \$5,000 per week. We will not include any travel or miscellaneous expenses that might be a part of a proposal of this nature. We will also assume that this is a small to medium sized project.

First, each phase can be priced individually prior to the project being started. The standard thinking behind the pricing of this project might be as follows:

- Analysis: \$20,000 since this is a small to medium sized project we will assume two resources for two weeks each.
- Design: \$20,000 same reasoning as before. We will need two analysts for two weeks each.
- Construction: \$45,000 we will need a developer for 6 weeks and an additional developer for 3 weeks. There may be some other costs involved here such as time for the analysts to assist in unit testing.
- Testing: \$10,000 two weeks of full system testing. This will also include time for the developers to fix bugs or errors found during the testing.
- Implementation: \$5,000 one week. Includes the development of the cutover plan as well as the implementation itself.
- Total project value: \$100,000

Some points of interest about this project: This pricing does not include any knowledge of the complexity of the project. It does not know how many business rules are involved. There are no project management costs in this estimate. Typically a project of this nature is a time and materials effort. This means that any phase is liable to go over budget for a variety of reasons. This requires a large amount of oversight by the project manager of the client to ensure that the project stays on budget. Since the original estimate does not

have any knowledge of the true complexity of the project the construction and testing estimates are purely guesses as to the level of effort needed to complete the project. The second method of estimating can eliminate this guesswork to some degree.

The second method breaks the project into two parts. The first part is the analysis and design phases. The same estimate would be given for those two phases because no new information is really known at this point to change how these phases would be estimated. The second part of the project is the construction, testing and implementation. The estimate might be changed for this phase of the project because more information is known about the complexity of the project after the analysis and design. This provides a little more security for the client because they feel like they are getting a better feel for the total cost of the project.

This entire process does not enforce any of the business rules of the new system in any standard way. This means that each programmer may code the same rule differently in different modules or that the rule may be left out all together. Data rules are not enforced in any standard manner. For instance a field in one table that relies on another field for its value may or may not get the correct value every time it is entered. It is purely dependent on the programmer of each module.

Code Generation via AppOnTap

Now let's examine how applications will be build using N•Cycles' App on Tap. We'll refer to App on Tap as AOT throughout the rest of this document.

The typical AOT project consists of three phases. The first two phases are the same as in a standard life-cycle project. That is there is still an analysis and design phase. The primary difference between the two approaches is that the AOT analysis and design is done using the AOT tool, the business rules are documented in the tool, and the screen designs are documented in the tool. The primary benefit of documenting using the tool is that once the analysis and design are complete 80% of the application system is already built. AOT utilizes the documentation stored in the tool to create the screens, data rules and validations for you. This eliminates the majority of the coding and testing phase. The tool creates code that works, and has already been tested to perform as expected. In addition, any changes made to analysis or design documentation are immediately reflected in the application. There is no re-writing of code. Now we certainly have not created a tool that can create code for any business system. That is impossible. We have, however, created a tool that breaks down the basics of any application into their components and eliminated the complex coding required to maintain those components. There will definitely be custom written code to provide the features that make this application unique to the business problem being solved.

You are probably asking how this changes the pricing of an application system. This is simple. First we determine the relative size of the project. This is done by asking how many different subject areas are involved, how many business rules may be involved, and by getting a good feel for the application. Senior analysts with experience determining the complexity of application systems perform this work. Most of N•Cycles analysts have 10 years or more experience in the software development industry. Once the complexity of the project is determined a price is given to the analysis and design phases. This typically is very close or slightly higher than a similarly scoped project using standard

methodologies. Let's assume then that we have priced the analysis and design phases of the project the same as the standard way at \$40,000. Here is where the real savings comes in. We then determine the price of the construction by the exact number of screens, tables, and business rules that must be coded. This portion of the project can be fixed priced so the client has a very high level of comfort with the pricing. Since the application has been documented through the AOT tool we are able to very quickly have approximately 80% of any new application up and running immediately after the design phase. The remaining 20% of the work is coding of business rules that are unique to the business problem and writing any new or totally custom screens that cannot be created using the tool.

One of the primary features of using AOT is that it forces the programmers into using a structured methodology and storage. AOT stores all it's information in an Oracle database. The screens are created directly from the information stored in the database. Data access is restricted to a data access layer (DAL) stored in the database on top of the actual table structures. The data access layer makes standard calls to enforce the standard business rules of the system. The DAL also makes calls to the custom code for unique business rules and other customized code. This is all stored in the database to ensure that all programmers execute the same set of code when storing, updating, selecting and deleting data from the database.

How AppOnTap Works

Let's first take a look at the architecture behind AOT. AOT is a JAVA servlet based application with an Oracle database backend. It uses industry standard components. AOT consists of three layers:

- 1) Business Rule Layer:** This layer contains all the standard business rules for the system. These are coded in PL/SQL (the procedural language built into Oracle) and reside between the actual data structures of the system and everything else. The purpose of placing this layer in the system is to force all data accesses and storage to utilize the business rules. The rules are written based on the tables in the system. That is each rule corresponds to specific actions that can be performed on a table.
- 2) Data Access Layer:** The data access layer, or DAL, isolates the data accesses from the remainder of the system. This ensures that the business rules are followed and that no invalid or incorrect data can reach the tables. Bypassing this layer and the business rules layer can invalidate the data model and cause incorrect business decisions to be made.
- 3) AOT Layer:** The AOT layer is a set of database tables describing the screens and components that are included in the system. The JAVA Servlet reads these tables to render the actual browser based screens that the end users see. This layer is highly configurable by the end users to provide both a customized look and feel for each individual user as well as an overall look and feel defined by the application owner. This provides great flexibility in the design of the application.

The following diagram depicts the different layers of AOT.

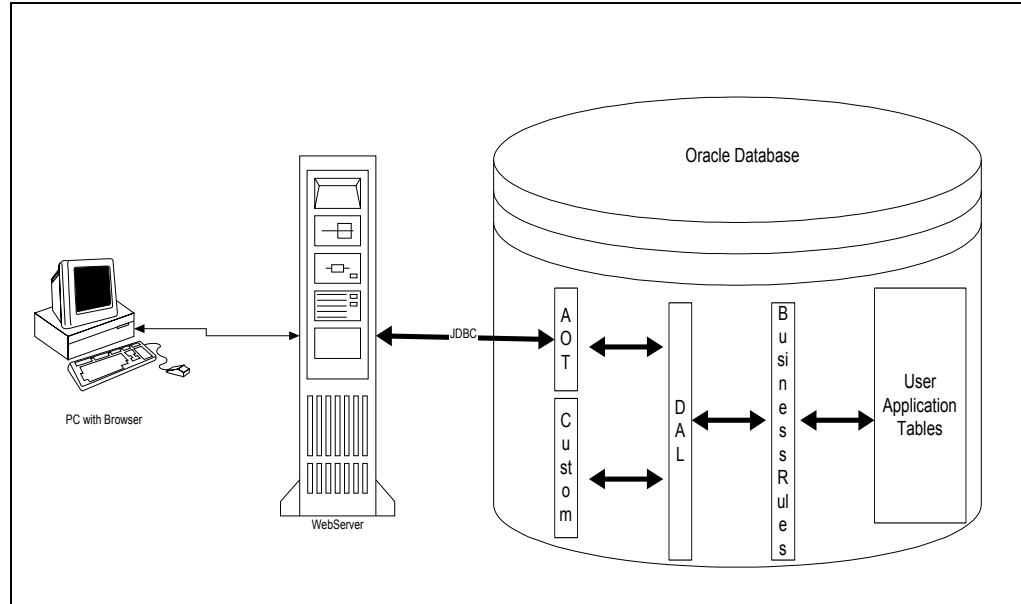


Figure 1. AOT Application Architecture

This application system will work with any WEB server that supports JAVA servlets. These currently include Apache Web Server, iPlanet Web Server, Microsoft IIS, IBM Websphere, and Oracle 8i JServer as well as many others. This allows our solution to work in virtually any IT environment.

As you can see from the architecture there is no code written to actually render the screens. This code has already been written and optimized. It resides in the servlet itself. What this means is that 80% of the code has already been written for your system. The code does not need to be tested or implemented. Once the screens have been set up in AOT they are ready to use. Simply point your browser at the AOT URL and you have instant access to your new system.

What's the Real Difference?

That's a good question. You are probably saying you must still do analysis and design the only difference is how the application is constructed. True, it is necessary to do analysis and design. In fact we feel that analysis and design is the most important part of the project. No code should be written until these two phases are completed and done properly. We feel so strongly about this that we only put our most senior resources doing analysis and design.

Once these two phases of the project are complete we have already documented and coded ~80% of our new system. This code does not need to be tested. It has been tested by N•Cycles. We know that it works. We guarantee that it works. The real work in the system can now be done. That is coding of the business rules and any customized screens that cannot be handled through AOT. Time does not need to be spent writing basic maintenance forms and data entry forms. These can be handled through AOT with

the click of a button. The real effort is in coding the parts of the system that are unique to your business.

Another major difference between the AOT method and traditional methods is that we can do more projects in the same amount of time with greater accuracy and customer satisfaction. Since the majority of the coding has already been done we can concentrate our efforts on the analysis and design. We make every effort to ensure that we have these phases correct. We feel that putting more emphasis on the analysis and design provides a better end result, and a better end result makes for a happy customer who is more likely to do repeat business with us.

With this approach we also use smaller teams. The ideal NCycles team consists of three people, one or two senior analysts/developers and one or two less experienced analysts/developers. Traditional application systems would utilize two analysts for the analysis and design phases and then a team of 3-5 developers to complete the construction. The client might not even be assured of getting the same analysts from the first two phases to assist in completion of the project. This means there is a lot of wasted time while new resources are brought up to speed on the project. There is a lot of overlap in responsibilities and a lot of wasted time. Using a smaller team promotes communication among the team members and allows us to rapidly bring less experienced staff up to senior level qualifications. The smaller team can also work quicker because there is less inefficiency in communication and work distribution. An ideal project using AOT is 8-12 weeks in duration. A similar project using traditional methods would probably last 3-5 months at a minimum. This is not including time spent revisiting an incorrect analysis/design. The moral of this story is that we can do more projects with fewer resources in a shorter amount of time for the same amount of money.

Advantages of AppOnTap

You are probably asking yourself why would someone decide to use this product over a traditionally implemented system. There are several reasons.

- Proven architecture. This architecture is used in many of today's prominent WEB sites including www.columbiahouse.com, www.nationalcar.com, and www.siliconinvestor.com, just to name a few.
- Industry standards. All aspects of this system utilize industry standard products, from the latest Java development kit, to the Oracle database, everything here is state of the art.
- Faster completion time. Average projects take one third less time than a traditional project.
- Strict data integrity. Both the relational database system as well as the business rules that are being enforced validates all data.
- Ease of maintainability. There is a single point of control. One user interface to learn to manipulate the environment.

- Portability. The system will run on any browser, and on any Web Server capable of supporting Java Servlets. This opens the system up to almost every possible combination of Web Server and OS platform.
- Consistent Interface. AOT provides a consistent look and feel to every application that is created. This allows the users to become familiar with how to use one system and can easily switch between applications.
- User customizable. AOT provides a way for each user to customize (to the extent the developer allows) each application to his/her own needs.
- Proven methodology. The methodology used to implement the system as practiced by N•Cycles is proven to work and provide quality systems.
- Insulates developers from low-level component programming. For example, Java servlets, PL/SQL etc. This means the developers can concentrate on the business logic involved in the solving the problem at hand.
- Single point of entry for business rule logic. This allows business rules to be enforced throughout the enterprise regardless of the end application that is accessing the data.
- The business rule architecture allows for the opportunity to perform impact analysis before a change to a business rule is made.

There are many other reasons to use AOT but these should cover most people's doubts and fears.