This draft was later published in Spectrum Magazine in 1996.

Ten Pretty Good Tips for Your First Successful Client Server Application

Chuck Thomas
ICON Solutions, Inc.

Here are some tips I've learned from the past six years of developing client/server applications across a variety of technologies for a variety of applications in a variety of enterprises. You can learn from my past miskates, er, mistakes, which I promise never to make again.

Before you can begin to develop anything useful, you must first ***establish a vision*** of what it is you are trying to create. In the context of client/server development, you must establish a vision of what your enterprise computing is to be...where do you want to take it? This is an enabling vision, not a vision of functional requirement. Basic stuff like do you want users to share data in a single monolithic database, or can they share through replicated databases? What will be the desktop business software suite (e.g., Microsoft Office or Lotus SmartSuite)? What will be the LAN backbone? What kind of groupware is needed? What environment is on the desktop, Windows for WorkGroups, OS/2, Win95, UnixWare, Windows NT, Solaris, etc.? What kind of database engine(s) do you need...large RDBMS like Oracle, SQL Server (either brand), or Informix, very large warehouses for EIS and OLAP, like Red Brick or Essbase, or smaller, like ACCESS? What kind of server(s) do you need and what environments should you support, like NT and UNIX flavors? What is the lifecycle for the legacy applications...which should you extend, which should you replace? Your vision for the enterprise IS is key to several points covered later, including the requirement for infrastructure.

> H.L. Menken said...
> "For every complex and hard to understand problem, there is a simple and easy to understand...
> wrong answer!"

I heard the following which I'd like to attribute, but I don't remember who said it:

Why Client/Server Is Like Teenage Sex

- It's on everyone's mind all the time
- Everyone is talking about it all the time
- Everyone thinks everyone else is doing it
- Almost no one is really doing it
- The few who are doing it are:
  - Doing it poorly
  - Sure it will get better next time
  - Not practicing it safely
- Everyone is bragging about their successes, though few actually have any

Perhaps no one said it and it was better left unsaid. Vote by dialing 1-900-DONTSAY.

Now this gets us to the Ten Pretty Good Tips[1], which are in the form of "Don't and Do" tips. Are you ready?

1. **Don't**...Assume that IS knows more than the user.

The customer is...

always the customer.

- The user always knows more about what they do
- The user always knows more about what they need
- The user always knows more about what they want
- The user is not limited by what can go wrong
- IS is there to enable the user vision, not vice versa

Remember there has been a fundamental paradigm shift...from the *supplier* of computing technology to the *consumer* of the technology. Unless you understand that in your organization the software crisis is not a technical problem, you haven't made the leap to understand that it is a management problem. We cannot provide fixed solutions to fixed problems, because there are no fixed problems. Flexibility is essential. We must design software for the ***virtual corporation*** of the twenty-first century. I bet you already know this, you just aren't sure how yet.

This is the age of empowerment of knowledge workers. This must be a part of your vision. They must have access to the global fountain of information. You can multiply the effectiveness of your Sales and Marketing group by providing easy access to the Internet; you can bring geographically distributed teams together inexpensively through ISDN connectivity and video conferencing and whiteboarding. Decentralize effective decisions by changing the "E" in EIS from Executive to Everyone's. Hell, give 'em all drill-down. Empower them and your enterprise will be powerful.

2. **Don't**...Choose a mission critical application.

Do something simple first...simple, not trivial. Make it a project of less than three months duration with fewer than four people. Make it something your users can live without.

Don't screw with the legacy applications! (More on this later).

Play with methods and with tools. Get familiar with the technology and with the mindset. Play a couple of tools against each other. Invest in a little software; I'm talking several thousand bucks, not several hundred. Like any craftsman, your software builders must have good tools to do good work.

Do the toughest and the riskiest things first. Projects fail because the natural tendency is to do the fun, pretty, and easy stuff first. Always begin with the high-risk parts, and remove the risk. If the risk can't be overcome, the project was ill-conceived and will die young, rather than have a lingering death.

Do it right. Get it working. Get user feedback. Then...throw it away! Your very first client/server application has too many compromises, shortcuts, mistakes, lessons, and support issues. It was a learning experience.

[1] The "Pretty Good Tips" idea came from a talk by Coleman Sisson of Powersoft, although the tips themselves are from my personal experience...don't blame Coleman.

Now do your first real client/server application:

- Develop a mix of mentors, craftsmen and apprentices
- Expect failure and washout
- Use a methodology
- Establish deliverables
- "Brick-Wall" the project...when you reach a predetermined point in time, it is finished; manage to that objective
- Measure against deliverables
- What is the application? Drill-down through the legacy data...
    - The data source is known and understood
    - Data moves uni-directionally from legacy to client/server
    - There is low risk and low visibility
    - There is high user satisfaction if it works
    - It is good integration and tuning practice

Continue the evolution of this initial application. Define and build a Data Warehouse. Start small, with a mini-Data Warehouse: a Data Shed. Make the changes evolutionary, not revolutionary. Practice making changes to the client desktop environment while the application has a small audience. You will need to know how to do this well. Continue the legacy integration, using but not creating data in the legacy systems.

3. **Don't**...Radically change the legacy applications.

Keep the risk of your initial projects low; make their impact high. Choose those things the legacy applications have difficulty with, and which high-level users have demanded. Don't make big promises.

Use the Data Shed approach. Don't pull data directly from the legacy applications, unless your legacy database is contemporary enough to support the long term direction of your vision. This means "open" and it means "SQL". It also means transaction processing, replication, smp support, and a lot of other things that are subjects for other articles.

Do not update the legacy applications (database/files). Initial client/server applications should "pull" data from the legacy applications, not push data to them. You should have to make zero changes to the existing applications.

4. **Don't**...Build up expectations.

The IS Professional's Most Feared Words...
The Programmer saying, "**I'm done**"!

- Does it meet the spec?
- Is it tested?
- Is it documented?
- Has it felt production volume data?
- Has it had end-user feedback?

Manage expectations. "Hope for the best...expect the worst".

- Deliver more than you said you would
- Deliver it sooner

- Deliver it cheaper
- Brick-wall the project...when you reach the wall, it is finished.

5. **Don't**...Establish the infrastructure as part of the project.

- Prepare and test the infrastructure before any applications are ready
- Alpha and Beta test the infrastructure
- Is the infrastructure everywhere the application will be?
- Remember, once it is on their desk, it is their desktop
- You may giveth, but you may not taketh away

Dig the hole before you pour the concrete.

What the hell is "Infrastructure"?

- The Network
- The Server(s)
- The Desktop Workstations
- The Development Environment
- The Business Suite
- Support

6. **Don't**...Always pick the "best of breed".

The best technology is not necessarily the marketplace winner, but the marketplace winner is always a good answer. The real question is, "Best at what?" Remember, "breed" is just another way of saying multiply. Once acquired and in use, its use will multiply. Can you continue to support it? Is it as good at production quality use as it is for development?

In acquiring tools, my philosophy is: Apples are always compared to Oranges. Choose the best:

- To fulfill the vision
- For the long haul
- To leverage technology for...
    - Object oriented design and reuse
    - Distributing data and processes closer to their source
    - Development efficiencies
    - Operational efficiencies

We must all respond to the ***custom application development imperatives*** for the 21st century. This means leveraging technology to solve business problems, for flexibility and responsiveness to the marketplace and to the business unit(s). We are in the age of the Virtual Corporation. It has no fixed size or shape, or place or even function. It merges and divests, grows exponentially and down-sizes, focuses on a single product and is all things to all people, is a products company and a services provider, is local and is global. Your systems must meet this profile.

7. **Don't**...Proceed without a methodology.

- Understand the requirement
- Formalize the design
- Rapid Application Development (RAD)...
    - Data-centric design

    - Iterative prototyping
- Deliverables-based quality review
- Reuse and more reuse

Methodology is the subject of an entire article. For now, just understand that for your first application, learn to data model, make the model as complete and perfect as you can in a given time period, and then develop a navigational prototype. This is a non-functioning hands on application that will present the look and feel and the navigation (from on window to another, menu traversal, and so on) of the real application. You users can see how it will play, and you get their feedback. You produce another navigational prototype iteration (presumably building on the first), get signoff, and then build one to three functional iterations. Finally, the testing and production cycle occurs. This is but one small part of a methodology, over-simplified.

A Brief Tale

Alice: Tell me, sir, which way I ought to go from here.
Rabbit: That depends on where you want to get to.
Alice: I really don't care where.
Rabbit: Then it doesn't matter which way you go. C.S. Lewis

8. **Don't**...Rely on the re-education of existing staff.

Everyone is a techno-bigit. We all know that what we know or what we know *about* is the "best". Visual Basic is the best. No, PowerBuilder is the best. No way, Forte is the best. Or it is Sybase, or Oracle, or Unidata, or Progress, or DB2. Or UNIX or NT? Or Windows or Win95 or OS/2? It used to be an argument about IBM or DEC or HP. Now, who cares about them? The worse person to decide on the technology is the implementor, who has a vested interest in what they already know, and not in implementing your vision. And these will be important decisions.

Everyone must be trained. Even Einstein went to school. It is not about having smart people on your staff. There is a tremendous amount of ***knowledge without understanding*** out there. So maybe they know the new tools, but do they know about architecture, infrastructure and methods? What about reliability and performance?

The best thing to do is to **rent** successful experience and then **steal** the knowledge. This means bring in some outside talent; not just any talent, but those who have done what you want to do. It doesn't make sense if you are training the consultant, too.

If you must do it yourself, choose a select staff to train...and these are not necessarily the "best and the brightest", because it ain't just technology. Teach them theory and methodology first; then teach them technology, the tools and languages.

9. **Do**...Keep your GUI simple.

Establish a set of Visual Standards (base these on the published Microsoft standards). Establish a visual framework, which is the enterprise application "shell", onto which other applications will be built. I would like to push you toward

object development and the use of base class libraries, but that is too much for this article.

Build the navigational model. Use the parent/child paradigm. Make your application MDI (Multi-Document Interface), which means that within your framework you can tile or cascade multiple application windows.

Make the design understated and elegent. Don't put too much GUI into it. Most first applications I've seen in GUI are garish, too loud, too much color, too much stuff on one window. Don't overuse graphics; don't be cute. Use a button bar to simplify user operation.

Use OLD/DDE to link seamlessly with the business suite of applications (like MS Office or Lotus SmartSuite). If you are showing a table of numbers, a click on an Excel icon should launch Excel, format the sheet and place the data on the sheet in a manner similar to the application. Or the application should be using a OLE to Excel in the first place, depending on what you're doing.

Storyboard the application. Just like in Hollywood, play at navigation on paper, walking through the application. Find the usability problems here, before your fingers hit the keyboard.

10. **Do**...Learn event-driven programming.

The difference between old-style procedural programming and event-driven programming is that before the programmer determined the way you did your work, and with event-driven, you determine how to do your work. Remember that the new application is not just a screen-scrape of an older application; it is re-thought, re-conceptualized.

11. **Do**...Learn relational technology.

The foundation technology of successful client/server applications is data-centric design based on the IDEF1X model, in which the logical data model is "normalized" according to relational rules. Here is how data centric design works:

Incorrect data models will create inordinate long-term development and support costs!

- First, build a logical data model, which is a normalized form of the Entity Relationships defining all the data within the scope of the application.
- Then, physicalize the model, which (yikes!) de-normalizes it to meet the efficiency needs of the database engine and application functional requirement. This means adding such things as "surrogate keys", which are numeric keys that replace long string keys simply to speed access.
- The modeling tool should be able to feed the application development tool (or is a part of it), such that many of the queries or data displays, triggers, and formatting structures are already known to the development tool, freeing the programmer from those concerns.
- If you change the data model, formally change it in the model, not just the programs and database.

Oops, I said there would only be Ten Pretty Good Tips, and there were eleven. That is only a ten percent overrun. Oh, uh, and...

12. **Do**...Plan to be incompetent.

    It is a learning experience. Draw from the successful experience of mentors, gain from the technology transfer. Plan for overruns. Plan to be late. Plan to recover from failure. This technology is new to you.

You can deliver successful client/server applications on time and within budget that meet and exceed user expectations. They will be your most flexible and information rich applications. Methodology can provide robustness, reliability and reduce cost through reuse...but that is another story.