1. *Describe the most difficult project you ever worked on and explain what made it so difficult.*
   The most difficult project that I have ever worked on was the quality request management system for Lumbee Industries. I contracted on this project to help with some architectural decisions and to do data modeling, business logic and UI development. The challenge of the project was that the project lead buffered all access to the client and had a tendency to hear what he thought was important rather than what the client thought was important. I tried to share some insights and experiences from having worked in a manufacturing environment and with quality improvement processes but these largely fell away to his conception of what was important for the project. A number of times we ended up with the wrong data model or the wrong navigation because the project lead remembered talking about a key aspect of the system but did not think that it was important enough to share as one of the goals.

2. *Given a specification for a task, how would you go about estimating how long it will take for you to complete the task?*
   Given the specification I try to analyze the pieces that are necessary to support the feature implementation – new data models, refactored source code, new or updated UI, etc – and build up time estimates for each based on past experience.

3. *Do you prefer open-source or proprietary technologies, and why?*
   I tend to prefer open-source technologies for two reasons. First, the communities around open-source technologies tends to be much more open and supportive than those that surround proprietary technologies. I have described the Rails community, for example, as primarily a group of very sharp developers who build great solutions while looking around and thinking that everyone else is smarter than they are. As a result the community tends to be far more congenial and deferential in its discussions and quite willing to give credit to bright ideas from others in the community. I've rarely posted a question that went without an answer for more than a few hours. By contrast, the community around .net tends to group into cliques of insiders who share only among themselves and hold others in contempt. The second reason that I prefer open-source technologies is simply for the source code itself. One of the greatest ways to learn new idioms and techniques is to read through well-written code. While open-source technologies do not guarantee well-written code it at least affords the opportunity to read and learn.

4. *Where do you think most software projects go wrong?*
   Most software projects go wrong by trying to design the end product from the beginning and consequently setting unrealistic expectations and developing unnecessary code. Without a question it is important to begin with the end in mind so that short term decisions can be made with the fuller context in mind. However, I have seen too many projects that focused on the "full product" with all its bells and whistles and then dove into developing the "sexy" aspects of the application with a poorly connected or haphazardly conceived infrastructure. Instead I have found that projects that take an approach similar to that described in *Getting Real* – focusing on a narrow set of requirements to solve a particular problem well and then letting other functionality "beg" for inclusion – have generally had greater success.

5. *Describe an experience working in a team. What was your role on the team? How did you communicate with other members of the team?*
   At PCSS I've been the team lead on the next generation of judicial solutions. In that role I

exchanged project status and high-level project goals with the company president in Texas primarily through email and phone calls. Attempts to have regular project reviews were difficult due to his travel schedule. The local team worked with a SCRUM-like approach, with brief weekly meetings to review accomplishments, discuss barriers, and choose assignments from the backlog. In addition to the regular meetings we often had impromptu whiteboard sessions to discuss how to approach a particular problem. With our offshore resources I conducted a second planning session, reviewing their contributions to the project and sharing new assignments. We communicated regularly through email but I pushed more extended discussions and code-sharing into our Basecamp account so that others could be involved and the results could serve as a guide for the future.

6. *What technologies would you choose to develop a high performance, highly scalable web application and why?*
   I would begin with Ruby and Rails. I believe that the expressiveness of the Ruby language makes the code simpler to maintain long-term. The integration and emphasis on TDD technologies helps to bring about better code and the MVC framework helps to provide clear separation of concerns. Depending on the needs of the system I would prefer either a Postgres or MongoDB database, or possibly a combination of the two. Postgres has some very nice master/slave solutions helping to reduce database bottlenecks for transactional systems and the potential to implement multi-tenant systems with true data separation. For systems or aspects of systems that do not require transactions MongoDB has a highly scalable key-value like store. Their “eventual concurrency” model emphasizes speed and availability that is important for applications with social content.

7. *Describe the most and the least rewarding experiences you've ever had while working on software.*
   The most rewarding experience I've had working on software was a small application that I developed with the children's minister at my church that helped him sign up and manage 500 children for a week of Vacation Bible School. He was a great project sponsor with a good grasp of what was possible and the patience to get there. When unforeseen needs arose we were able to quickly specify and deliver a solution. Those who used the system were thrilled with its responsiveness and usefulness, particularly in comparison to the product they had used in previous years.

   The least rewarding software development experience I have had was the three years I spent developing the EZLINK product for the Dodge division of Rockwell Automation. This was a greenfield project in which I developed the firmware for an industrial monitor and had some input into the hardware decisions. The difficulties with the project were manifold. The company's pricing model made it difficult to select components that were a good fit for the application. By tradition the company had a very “steel and concrete” mentality, making the integration of electronics and monitoring a difficult fit. Despite the product winning an award it was not included in the company's own list of growth products.

*What's the latest thing you've learned, or what did you learn last week, or what are you currently learning?*
When are you *not* learning?! From a technical perspective I've been working more with Cucumber and looking to find best practices in rspec. I've also been getting to know git and holding hands with

Erlang.  At home I'm constantly trying to learn more about my kids so that I can connect with them and about how to be a good parent so I can lead them well (and hopefully learn from my mistakes without their own pain).  I am also trying to learn more from some of the great soccer coaches around me in my soccer club so I can help the boys on my team develop to their fullest potential.  I am always looking to learn more about my faith both for the rewards of the future and the impact on my relationships and service to others today.  For me, life is constant learning.  I will not master it all, but I will enjoy the process along the way!