



What Proficiencies are Needed For a Software Architect?

By Cliff Berg

A software project's architect is the project's most critical technical decision-maker. A software architect therefore needs to have superior technical judgment, and must be skilled in assessing tradeoffs and managing risk. However, a software architect also must have broad people skills, to serve as an effective mediator between and among software developers, project management, and stakeholders. To be effective in this role, a set of skills and knowledge are needed, a portion of which may be innate in some architects, and others which may be learned through experience and training.

I recall in 1995 seeing a job posting for someone with "at least two years of Java experience". Little did the recruiter know, Java had just been announced by Sun and the only people who fit that criteria were those on the small team at Sun who had developed the original Java language and APIs.

Most organizations that hire software development staff tend to recruit based on the wrong criteria. Requiring "two years experience" with certain APIs is one of the most notoriously inappropriate requirements. Every programmer knows that

when any project starts they will be familiar with only half the APIs and technologies and will have to learn the rest through self-instruction and experience. This is a never-ending game. In fact, experience with APIs is one of the least important criteria: rather, experience in solving certain kinds of programming problems, and experience with the associated issues and risks, are far more important, and these cut across languages and APIs.

Inappropriate recruiting criteria is more predominant the higher the software position. For an entry level programmer, one prerequisite is surely experience in programming. But what about a senior programmer? A technical lead? An architect?

Merely knowing how to program is wholly inadequate as a skillset for a software architect. If you add experience with the technologies that are to be used on a project, the skills are still inadequate. If

you throw in knowledge of design patterns and object-oriented design, they are still inadequate. What about familiarity with the ways in which

infrastructure impact an application? What about all the "ilities", that is, reliability, manageability, and maintainability, not to mention security? And this list is just for starters.

MERELY KNOWING HOW TO PROGRAM IS WHOLLY INADEQUATE AS A SKILLSET FOR A SOFTWARE ARCHITECT.

Architect as Decision Maker

Martin Fowler [Fowler2003] has made the point that software architecture is not a set of diagrams. Rather, it is a set of decisions. If a software architect is the most senior technical member of a software project, then it stands to reason that the decisions made by the architect are the most important and difficult technical decisions; and if architecture is all about decisions, then it follows that above all the architect must be the best technical decision-maker on a project.

So the question of what an architect needs to know can be transformed into the question of what someone needs to know in order to make the best technical decisions on a software project. Everything else is secondary.

When I was the CTO of a software development company, I used to tell our programmers that it didn't matter what kind of technology a system is built with, that the issues are the same, and what matters is that all the important issues are addressed in an acceptable manner. Different technologies provide different means of addressing issues. For example, EJB and Web services are different technologies for implementing services, and each provides different APIs for service lookup; but the need for service lookup is still there and must be addressed.

**THE DECISIONS MADE BY
THE ARCHITECT ARE THE
MOST IMPORTANT AND
DIFFICULT TECHNICAL
DECISIONS.**

In order to make a good decision about how an EJB application or a Web service should be designed, one must be familiar with the issues associated with remote services, such as service lookup. One must also be familiar with the features and APIs provided by the technology, whether it be EJB or Web services or something else. If one is familiar with the issues, it is usually fairly easy to research the features of any given technology, look for how those issues can be addressed, and thereby learn enough in order to be able to make good decisions. Familiarity with the core issues puts you on guard for what to look for. On the other hand, if one is merely familiar with the technology APIs, the issues might not be apparent, and it is possible to misuse the technology.

Familiarity With Technology

Familiarity with the chosen technology is important however, even if it is less important than familiarity with core issues that cut across technology. Familiarity with the technology allows an architect to make decisions more quickly, and to

be sure that he or she is not overlooking an important weakness in the technology that is not immediately apparent from documentation. This is where a good technical lead comes in: someone who is knowledgeable about the chosen technology. Such a person can advise an architect about the idiosyncrasies of the technology. Of course, it is more efficient if the architect can rely on his or her own knowledge, but one cannot know everything, and if it is a tradeoff between an architect who knows the issues and one who knows the technology, I would rather have one who knows the issues.

Technology is not merely APIs or a chosen software platform. It encompasses every tool that is used in a software project. This includes languages, compilers, debuggers, integrated development environments (IDEs), application servers,

databases, and so on. Further, issues encompass

IF IT IS A TRADEOFF BETWEEN AN ARCHITECT WHO KNOWS THE ISSUES AND ONE WHO KNOWS THE TECHNOLOGY, I WOULD RATHER HAVE ONE WHO KNOWS THE ISSUES.

not only those considerations that are immediate to the project's problem domain, but also fundamental

issues such as the motivations ("forces") behind chosen design patterns and algorithms. Such issues reach back to fundamental computer science.



Communication

All this knowledge is useful to make decisions, but decisions are useless if one cannot communicate them. If a software architect is a project's senior technical decision-maker, then that person must retain a high level of de-facto legitimacy among the technical staff. Otherwise the person will become sidelined by those building the system. This happens easily because in a software development environment, knowledge quickly translates into power, and software developers

have the most intimate knowledge of the application details. If the architect cannot build their respect, then he or she will lose credibility and will suffer from reduced effectiveness. The ability to communicate with and lead a technical team is therefore crucial. In addition, an architect who is a good mentor will succeed in transferring his or her knowledge and skills to the rest of the team, whereas one who is not a good mentor will merely become an indispensable “guru” who leaves a vacuum behind when moving to a new project. Therefore, mentoring skill and the tendency to delegate are important for an architect to have.

The ability to communicate effectively with the project manager and with business stakeholders is also crucial, because otherwise the architect will not be effective at learning the concerns of the business, and will not be effective at helping the project manager explain to the business how the application will meet their requirements.

An Architect as a Bridge

An architect who can serve as a “bridge” between the technical team and the business is able to advise the project manager on the tradeoffs between software design choices and requirements choices, in order to maximize business value. To add value to such tradeoff decisions, one must be able to think in terms of financial and business tradeoffs, and be sensitive to the project managers concerns for schedule and cost. Such business-level decision-making is just as important as technical decision-making.

AN ARCHITECT MUST BE ABLE TO THINK IN TERMS OF FINANCIAL AND BUSINESS TRADEOFFS, AND BE SENSITIVE TO CONCERNS FOR SCHEDULE AND COST.

Summary

A software architect needs to have good technical judgment, rooted in years of experience dealing with design issues that cut across a variety of technologies, and knowledge of computer science fundamentals. An effective software

architect benefits from having knowledge of the technologies that are to be applied in a project, but this is not essential.

An architect should also be a technical leader who maintains the respect of the development team. The architect should be adept at mentoring others on the team, to help them grow into architects. An architect should be able to think in terms of technical tradeoffs as well as business value, and be adept at communicating with the project manager and business stakeholders.

References

Fowler2003 "Who Needs an Architect?" by Martin Fowler, IEEE Software, July/August, 2003, pp. 2-4.

