Feature Articles

TopCoder @ Work: The Hacker vs. The Architect

By **timmac** & rhudson
TopCoder Members

In an IT world where the names of the hottest new design patterns and development methodologies are thrown around like umbrellas caught in tornadoes, a competent application architect commands a certain position of power and admiration. Typically cast as a project technical lead, or similar role, this position of power is well deserved; today's increasingly enormous and complex applications demand a level of organization and sensible assembly, lest they should eventually outgrow the abilities of those who manage them. A good architect is like the sheriff in town, keeping everything in line and maintaining a good sense of order. Yet amidst all the talk of things like release schedules, factory patterns and deployment packages, an important team member of yesterday is now all but forgotten: the hacker. This is the more renegade individual--the cowboy who jumps on the back of his horse and chases down the thief. Although seemingly disorganized in a sense, he gets his job done in a way that makes sense and works well for him.

Now, this is not to say that most developers don't have some qualities of both the architect as well as the hacker, nor is it to necessarily stigmatize one over the other. However, for the sake of comparison, we will look at the two extremes using two fictional characters, Archie and Hack.

Archie has been a developer for seven years and was with his first company for a while. He comes from a background of writing GUI-based desktop applications and in the past two years or so, since coming to this company, has been working in web applications. The shelf of his cubicle includes books on database design, user interface styles and a fairly recent addition simply called "Agile Development For You." His walls are plastered with database schemas, flow charts and calendars with release dates circled. Archie thrives on risk management and structure. Sometimes he's open to new ideas, but only after careful review of how they fit into his formal traditions.

Hack sits a few cubes down from Archie. He, too, has been programming for about seven years, though he has been with several companies in that time, each for only a year. Hack considers himself to be a problem solver and once he's found the "answer" he loses interest in any sort of structured, standards-based implementation. That's boring. Hack would rather feed his attention by coming up with fun new ways to solve traditional problems and leave the details - and the cleanup - to someone else. His unplanned bursts of inspiration and unparalleled creativity often leave things like structure and specifications rendered irrelevant.

Hack is smart--very smart--but has low tolerance for the drudgery of repetitive coding tasks. Rather than spending an hour typing in field names from a database, Hack would rather spend four hours building a utility that would parse the database, determine the field types and print out standard template-driven code to do the one-hour task. This "optimization" can be an asset, assuming that Hack's utilities are always going to be re-used over time.

Archie appreciates Hack's work and has even installed a few of Hack's "power tools." But super utilities don't always improve deliverable time and Archie's clients depend on regular updates to the accounting software to account for legal and policy changes. These time-sensitive changes simply can't take hours or days or weeks longer than the wildly-fluctuating estimates specify.

Archie and Hack only sometimes get along. Whenever a change request is passed down from management, they offer wildly different estimates for completion time. Archie usually offers a date in alignment with an upcoming release cycle, while Hack will speak in terms of days or sometimes merely hours. Hack likes to give Archie a hard time about how long it takes for him to release a simple text edit change. Archie ridicules Hack for making code changes directly inside a live production system.

Hack's first position found him maintaining some legacy command-line based applications and other positions over the years have given him some exposure to applications on several other environments and platforms. His cubicle has no books at all, though he makes up for that with links on his desktop to many of the more popular API references. The only thing on his wall is a

poster for an obscure band, which nobody seems sure how to pronounce.

So, which one of them is right? Well, probably neither one of them. A good developer is both Archie and Hack as the situation demands. Let's consider how to reconcile the skills of the orderly sheriff and the reckless cowboy.

To marry this odd couple, Archie might consider working towards an agile methodology that creates smaller deliverables with more frequent releases. Hack then only has to work with a smaller part of the overall system and is free to exercise his "creative license" in ways that represent lower relative risk to the overall scope, which pleases Archie.

Archie can put his architectural skills to work by dividing larger modules into smaller parts and defining use cases that can be accomplished in shorter periods of time. This is not unlike the TopCoder process whereby projects are broken into components before being presented to the community for design and development. When Hack is given these tasks that range in terms of hours or days rather than weeks or months, the overall risk is smaller and even in the worst case of Hack not getting something done, the effect to the bigger time line is mitigated.

As we have discussed in a previous article about giving good time estimates, it is very difficult for anyone to truly estimate the level of effort involved in project tasks that range into weeks of work. This is particularly true for more general tasks that have yet to be clearly defined on a more granular level. For example, consider a project that involves writing an accounts payable module. While general business exposure might be enough for most employees to at least know what that means at a high level, it is doubtful that a developer could accurately gauge what that really entails, particularly if they have never been served that exact type of project in any of their previous ventures. Rather, once the architect and project manager put their skills to work by itemizing the specific tasks, our cowboy friend can be tasked with writing the routines to post cash payments, for instance.

Obviously, we have discussed mostly the extreme cases here, but it helps to further the point. Different approaches to development each have their merits on some level, but it is only when the best of each is used in the appropriate manner that truly productive results emerge.

Home | About TopCoder | Press Room | Contact Us | Careers | Privacy | Terms
Competitions | Cockpit