Architects Dont Code

"The SystemArchitect responsible for designing your system hasn't written a line of code in two years. But they've produced quite a lot of ISO9001-compliant documentation and are quite proud of it."

Name: *Architects don't code*

Problem: A large, complicated system needs to be easy and quick to build, debug and maintain.

Context: The development organization has a list of supposed junior programmers and a list of OO or relational "experts", such as senior programmers or solution architects.

Forces: Supposedly, having an expert architect will bring about consistency, cleanliness, modularity, and other characteristics of efficient software development. Expert time is expensive or rare or both.

Unfortunately, design and coding are two sides of the same coin. You cannot design if you cannot code and you cannot code if you cannot design. TheSourceCodeIsTheDesign.

Supposed Solution: Some expert, *usually* an authority, will architect the system on paper. The expert(s) won't be unduly bothered with technical details because their time is valuable. The team will be required to follow the architecture so that the good qualities of the system are assured.

Resulting context:

A project where the code reflects a design that the SystemArchitect never thought of because the one he came up with was fundamentally flawed and the developers couldn't explain why it was flawed since the SystemArchitect never codes and is uninterested in "implementation details".

Implementation lags and fingers are pointed; whoever has the most authority wins, usually the expert. The programmers may need to be replaced if they are deemed not to be "team players" because of arguing with the architect.

Real solution: Get architects involved at an implementation level. They will scream and moan but it's for their own good. They don't need to be writing (necessarily) an equal share of production code (if, for example, your organization is very adamant about the RoleOfASystemArchitect) but they need to get their feet wet from time to time and need to be aware of how changes in their design affect the project.

Alternately, eliminate entry-level programmers (the rank, not the employees) in favor of having the system only coded by those with enough knowledge (the MythicalManMonth solution).

Faulty belief: SystemArchitects don't need to code anymore. They know what they are talking about by virtue of the fact that they are SystemArchitects.

There is a sister AntiPattern called ArchitectsPlayGolf. -- SunirShah

Is this not simply a specific instance of RequirementsTossedOverTheWall?

If architects don't code, it's easy for them to get so distant from the reality of implementation that they fall into ClassicOoAntiPatterns, like FearOfAddingClasses.

I think that making architects write code is a valid solution.

I'm an architect. (I prefer to call myself a "technical lead".) I know that my designs are good because I'm usually the main person to follow through and implement them. *At which point, I find out how not-so-good my design really is, so I change the design during implementation. ;->*

The main problem (AntiPattern) I find with having architects/technical leads write code, is that they need to learn how to delegate work to people who aren't as skilled as they are. Top technical people have a bad habit of trying to do all the work themselves, while "lesser" developers go idle. "If the developers working for you aren't good enough to do an adequate implementation of your design, then ***it's your responsibility to train them!!!***" -- JeffGrigg

"Sorry, I don't have time to train the developers working for me, I'm too busy doing all the programming." See the vicious cycle there? Believe me, I know from painful experience that it's very *hard to break.* -- MikeSmith

I HaveThisPattern but I'm not sure exactly how I'm supposed to train these people. I have people doing some PairProgramming which is great and I also try to TrainByExample but it's still really hard to communicate certain concepts to people.

Can they read books or read papers, articles and documents in general? I have the same problem, they refuse to read anything, only code in the project and actively code in the project. When doing pair programming they look through the window, their cell or whatever, they don't pay attention, and all they do is hack the system

I think that the role of the *technical lead* should be similar to that of an XpCoach. The idea is to guide less experienced individuals so they see what makes a good design (and a good implementation). In order to do this the

technical lead needs to spend a good deal of time explaining as well as showing (by coding for instance). These two activities combine well in the PairProgramming practice.

Too much delegation leads to an unclear and fractured view of the project. Those involved must keep their level of knowledge of the project as high and comprehensive as possible. This means that the architect should be working with the developers and watching the evolving codebase to make sure that it corresponds to the design (of course if the developers do their own design this whole AntiPattern does not occur).

-- IainLowe

Do not be afraid to fire/suggest to leave/move. There are lots of reasons why people cannot work together. My personal productivity is 500% better, when I am among the people, who share the same ideas and approaches I value. It does not mean, that other approaches are wrong ones. They are just different. There is enough room for everybody. -- KonstantinIgnatyev

---

Last week, I had a job interview with a successful international consultancy. They regard ArchitectsDontCode as an ideal to be striven for. They are also very proud of their (proprietary) n-phase waterfall methodology. I asked if maybe it wouldn't be a bad idea to do some kind of prototype up there at the front. "That would be a waste of time," they said. Later, I was mulling over this situation, whereby this technical community considers another communities best practice (I emphasize that they are a *successful* consultancy) to be an anti-pattern. I don't know what conclusion to draw yet. -- KeithBraithwaite

(ps, I walked out of the interview.)

This happens all the time. You can't have all people in the world agree on everything, it would be dull, boring, and counterproductive, because no new ideas would be possible. -- GuillermoSchwarz

Depends on what they are developing. If I'm developing something I've done before, there's really no need for a prototype. New and/or risks points require prototyping. I hoped you explored what they were developing before hopping on your high horse?

In my experience, no project has been so similar that when starting it there was nothing new to experiment with and learn about. If challenging their assumption that building a prototype would always be a waste of time constitutes climbing upon a high horse, so be it. I get a better view from up here. -- Keith

Keith, you've been lucky. There are plenty of companies where they spend lots of time on projects with very little new to experiment with and learn about. These are the sorts of companies that tend to love having ChiefArchitects, in

part because the actual work is so bloody dull that they can't hold on to intelligent programmers.

In that case it still sounds like he was right to walk out of the interview, even if it was only because he doesn't want to work for a company like that. I dare say that some of the most successful consultancy companies do almost nothing new on each contract. If you want to make money, a really good way of going about it is to find something lots of people want/need (or you can make them think they want/need), find out how to do it and then do it over and over again till you can't make anybody want it any more. Then repeat.

Consumable items, or something constantly requiring or needing upgrading. A plastics company might sell the same old plastic for years, and make more than some company selling reusable plastic. So just because something is interesting and more clever(reusable plastic), doesn't mean it will make more money. Example: people buying new plastic sandwich bags each day is - great for the plastics company. Profits, good. - But could they have used zip lock sandwich bags? Reusable. Clever. Bad for plastic company. No profits.

Or... I can make 5 more dollars per box if I pack the box with chemical compounds (plastic, say). I will lose 5 dollars per box if I use cardboard, which is more decomposable. But, if a volcano erupts, that causes more harm, relatively, than any cardboard or plastic I was ever worrying about.

What is more important... money, long term, cleverness? Define cleverness! You can define money, but you can't define cleverness! That good or bad?

There are (may be) an unlimited amount of ways to look at a situation and analyze it though, and I am caught here rambling. However... if the item is free, are there in fact less worries in all aspects of the situation? I don't need to buy plastic bags each day, because they were free. I spent less time worrying about the plastic, since they were free, and since they were free, I chose the zip lock, because I spent less time worrying about the profit, or loss, or hassle. With free, comes cleverness.

Also, ArchitectsDontTest.

This is recognized as an AntiPattern in other fields of endeavour too. From the BasicOperatingRulesOfLockheedsSkunkworks (for groups that design new aircraft):

> *The contractor must be delegated the authority to test their final product in flight. They can and must test it in the initial stages. If they don't, they rapidly lose their competency to design other vehicles.*

Umm, actually, this is true. Architects *don't* test. They design architecture.

I think what's wrong with ArchitectsDontCode is that it's worded as a prohibition. The point is, architects don't have to code, as long as they have

ways to get concrete feedback on their ideas. I had an experience like this last week. I got an idea for how to use existing "hooks" in a database application to do something new at low cost. I worked with a "coder" to test the concept. He contributed some important tests I wouldn't have thought of. On the other hand, he built the application several years ago, and never got the idea I came up with. The perspective of someone who spends little time coding can be valuable. The prohibition angle just goes too far. I write code when I want to, but in this case writing my own code instead of involving Henry wouldn't have been as good. -- WaldenMathews

You're right, maybe it should have been ArchitectsWhoDontCode instead. I agree that a variety of perspectives are what gives a team its strength. My objection is mainly against those so far from the rest of the team that they lose site of common goals. -- IainLowe

---

Our Java development team has just received this email from our French project manager and software architect: "*According to a discussion involving several project leaders, it is preferable to use only one class BusinessRules.java. Consequently, we only need one function GetData, which is composed of a sequence of if statements (as a big select statement).*"

Additional information:

- the "big select" refers to the "select" statement in Visual Basic.
- this directive changes already written and working code where business rules were classes that implemented an interface (BusinessRule). Now they are blocks of code in if-statements.
- the GetData function is meant to be static and its responsibility is to add or modify fields of a RDBMS obtained resultset. Given the name of the field and the data in the current record it returns the value for the named field of the current record.
- the PM refuses to code, stating that his job is to draw the big picture and we're meant to fill in the details.
- the PM claims interest in XP

I'm really interested in your comments, so please make them even if all this paragraph gets refactored out later.

-- AnonymousToProtectTheInvolved

Ignore the suggestion. Either let them make the changes themselves or make them justify the change and your time to make it. Ultimately, you are responsible for the system, so you get to make the final decisions. Architects aren't programmers, so their opinions (and jobs) are useless.

-- Can't do that, he is also the project manager, I am merely a programmer. The cute thing is that the whole team voted to keep the OO approach, he accepted, then a few days later he sent this email.

If he won't code, then enforce it - the select statement is code, so you can ignore it as a figure of speech, or you can tell him to stop dabbling and really code. Basically, he either stays out of implementation, or he makes it work.

Vote with your feet. Fast.

Consider also: what subsequent changes will your architect's successor make? (DiggingAndFillingHoles)

The fact that it's possible to make a bad design work, through heroic effort, doesn't make it a good design.

Actually he and the PMs he mentions he talked to think this is good design. What do you think? I brought this into the discussion because it seems this is THE place to find some higher authority in software design today. Also, I think it is a good example of architects that don't code behaviour.

It's hard to come to a firm conclusion about the "goodness" or "badness" of a design by looking at only one method of one class. But generally speaking...

- Static methods are generally not object oriented. Now if you ***want*** a procedural design with passive data, let's be open and honest about it.

He is open and honest about it. When I made the argument that his design is not OO, he said it doesn't matter, OO is not a purpose in itself (which I agree). But he tries to do a completely procedural system using Java, which lately became really fun ;). Java was chosen only for platform independence.

- Case statements are generally a strong indicator that you're not looking at an object oriented design. Polymorphic dispatch to object instances generally eliminates the need for lots of if and select statements. Proponents of XP using Smalltalk have made fairly strong arguments that all if statements should be replaced with polymorphic dispatching. *(Reminds me of a discussion that started on NullObjectPerformance, which unfortunately doesn't really talk about polymorphism over case.)*

- As for "GetData" as the name of a function - I'd be hard pressed to come up with a more perfect example of a bad function name. It doesn't communicate anything. Can you think of any function that couldn't be called "GetData?" (Related idea: DontNameClassesObjectManagerHandlerOrData)

-- JeffGrigg

Master object-oriented programmers know when it's ok not to be object-oriented. Architects typically aren't master programmers, which is why they've retreated to boxes and lines. Connect the dots.

> My *definition* of Architect is Master Programmer. Anyone holding the

> title Architect who isn't a Master Programmer is a joke, to be mocked and ignored. That holds too for former masters who haven't programmed in years - experience fades quickly with lack of daily practice.

Right. I'd ask them to justify why they've abandoned widely recognized OO practices in this area; what's the business justification? Typically they can't - because they thought they were doing OO - and promised it to management. -- jtg

The justification is that OO in this specific case leads to having many classes (which would be harder to maintain than a big class with a huge if) and to many object instances (bad performance).

Measure the performance. I bet that the performance of the system will be the same either way. It is possible that having many classes will be harder to maintain then a big class with a huge if. How often will that class be changed? Will there be several people changing it? If two people are trying to change it at the same time, their edits to the huge if will conflict. However, you can easily make subclasses in parallel. But if you only expect two or three changes a year, this doesn't matter. But since most maintenance programmers who learn Java don't understand OO very well, the architects might be correct in stating that it will be easier to maintain a huge if. -- RalphJohnson

I believe the above is a good example of the problems with ArchitectsDontCode. When you have architects who don't code, they do not have a common frame of reference with those who do code. The architect may or may not have good ideas, but he cannot communicate them to the coders. Likewise, the coders may or may not have good ideas, but they cannot communicate them to the architect. The result is often that based on force of personality, bad ideas can be implemented and good ideas ignored. -- WayneMack

FearOfAddingClasses:

> *"The naive belief that adding classes to a design makes it more complicated."*

I think the bigger problem here is ArchitectAsManager.

Or ManagerAsArchitect? We have a project manager who has a technical background. He doesn't write code, but he used to and he remembers the freedom. Every once in a while he provides us with some really valuable insight on how our code interacts with our database (for example). But this does not make him an architect and the final decision is always up to the programmers.

Right: A good architect or manager has to empower the people who work for them: The workers need to be responsible for their work.

What PointyHairedBosses can't realize is that you can't empower people just by walking around saying "from now on, I'm going to hold you responsible for your results; you're empowered." You actually have to give them **decision making authority.**

Example: Some time ago, I had a programmer who was redesigning an interface to a DLL, changing it to COM. As his manager and as the technical lead, I encouraged him to expose more object oriented interfaces, rather than keeping with a procedural style. "But it would work if I do it this way," he kept objecting. "Yes, but there's a better way," is what I kept saying. Ultimately, I let him pursue the more expensive-to-maintain procedural style - I had to give him ownership and control of his work. -- JeffGrigg

---

moved from ChiefArchitect

It's funny that people nowadays have such a bad reaction to the term *architect*. Personally, I think we lose a lot more by being uptight about the architect word. Architect is *just a word* no different from other labels such as Coach, TeamLead, Mentor, SeniorEngineer, and so on. It just has a slightly different emphasis.

Personally, I don't think I would use the *term* ChiefArchitect, but I definitely believe in the role described above and have no problem with the word **Architect**. Also, I'm fine with only having *one* Architect and a DesignTeam composed of subsystem leads and/or designers. *But I am equally happy* with a project having anywhere between 1 and 3 architects. More than that and, I think ConceptualIntegrity could suffer. BTW, before people get upset, this doesn't mean that a system couldn't have many more designers and subsystem leads. I've simply been on more projects that lacked focus than those with a consistent point of view.

I think the architect word is a *good* word. In fact, I consider myself to be a SoftwareArchitect and aspire to be a better one. Nowadays, however, people seem to be really uptight about this word. I just hope they never get uptight about the word ProjectManager or Tester - then we'd really be in trouble. -- RobertDiFalco

For some reason, some wikizen has moved my comments above out of context. I do not believe in having an architectural role that does not include coding and never have. What I was referring to was the role of ChiefArchitect. I'm not sure why ChiefArchitect comments were added to a page talking about the anti-practice of designing but not coding. -- RobertDiFalco

---

I see a topic called ManagersDontCode. Shouldn't that also be considered an AntiPattern? I say this based on some recent horrible experience. -- BernardFarrell

---

In my experience (as also indicated above), if you don't code, you can't design

and also, if you forget your coding you produce designs that are almost not implementable or implementable at a very high price (performance and otherwise). In my opinion, an architect should take responsibility for the critical infrastructure, on which the success and failure of the project depends. The architect must make sure that this part of the code works and works in the way it is supposed to. To realize this, more often than not, an architect must sit with the developer, explain the concepts and **look over the shoulder** (not often appreciated by the developers) to get the code moving, if required, extensive coding may be also required on the architects' part. The point of the matter is that though (regrettably) companies exist that promote the **culture** of **Architects Dont Code**, in the successful ones (which intend to stay that way), **Architects Do Code (as required)**. -- Sanjay Debnath

---

There's also a variation of this where ArchitectsCantCode. Usually, this is because the architects are people who have expertise in the business domain or in some other technology like C/C++/Cobol/etc or network infrastructure rather than the implementation technology.

From MythicalManMonth (paraphrased from memory): "An architect should not specify *the* implementation, but should always be able to specify *an* implementation."

Also from MythicalManMonth: "As Blaauw has said, 'Where architecture tells *what* happens, implementation tells *how* it is made to happen.'"

So the answer to the question "How can someone who never writes a line of code be responsible for how that code will be written?" is "they aren't". Architects are only responsible for what the code does, not how it is written. If they start interfering with how the code is written, they become engineers instead of architects, and as engineers, they should code.

*Brooks uses a slightly different definition of "architect" than is supposed in this discussion. An "architect" in MythicalManMonth is responsible for **requirements**, not what is commonly thought of as "architecture" - although Brooks heavily bases his experiences on the building of OS's and the like - software systems designed to be consumed by software engineers; so the dichotomy is a bit fuzzy. But if you use the definition of "architect = system architect = responsible for requirements", then the what-not-how rule applies.*

In most shops, the "architect" goes a step (or more) down into the implementation - dictating high-level system design. When the architect does that, then ArchitectsDontCode can indeed be an AntiPattern - especially when the architect specifies a design that is difficult to implement, and the implementor is blamed for its subsequent failure.

Architects should held be responsible for (i.e. required to answer for) whatever they are responsible for (i.e. in a position of authority over).

Isn't this AntiPattern just a specific case of a much more general AntiPattern where someone performing a role that is considered to be "more valuable" doesn't get to experience the pain of someone performing a "lesser" role further into the development methodology (whatever it might be)? The "more valuable" role is typically seen as being earlier in the development cycle (certainly in WaterFall), requiring greater experience, being more costly to resource, and having some kind of authority. The "lesser" role is correspondingly seen as later in the development methodology, being less skilled, correspondingly cheaper to resource, and lacking authority.

Classic examples might be DesignersDontTest (resolved at least to some extent by UnitTest), and DesignersDontSupport (the pain of support staff is never experienced by the engineers who developed the product). In each case the underlying problem is an OpenLoop process where there isn't the required feedback to enable individuals to improve their skills.

I see two underlying forces that lead to the problem:

- Firstly, the organization is not prepared to let costly resources spend their time doing tasks that could be performed by cheaper ones.
- Secondly, some individuals feel that certain tasks are (now) beneath them, and refuse to get involved, or even listen to, those performing the tasks. This causes understandable resentment.

I tend to fall foul of the first, so have to find the time to test architectural ideas with actual implementation, or at least find out from the designers what issues were encountered.

The ideal solution has to be to move to a ClosedLoop approach, where feedback from every stage of the process is key. Practically, perhaps the best to hope for is some kind of formal feedback, rather than actual experience of the pain.

It would be interesting to compare if Building Architects are better or worse for not having handled nails, trowels, fork lift, etc. - the tools and skills needed to implement their designs. Did Frank Lloyd Wright actually build his designs? I suspect he built models but not whole buildings or the furniture in them which was part of his style - to have a holistic design from macro to micro levels. Or do composers know how to play all the instruments in the orchestra? Once you reach a certain scale in design (any field) there needs to be a separation of roles and it is not possible for the prime visionary to have detailed skills to implement the lowest levels. They require a different skill set.

I don't know what you mean. Good orchestra directors can (and do) play a lot of the instruments. I agree that the players may be better, but they HAVE to know in order to be able to direct. A person that has a degree in music (like the director) must be able to play chord, wind and percussion instruments

(and sing). Let's hope that one day architects on software projects get to the same level.

- Er...no. The analogy may be good, but the facts are inaccurate. I'm a software developer with a background in music, and I can assure you (at least in the US) that the requirement to play an instrument from each family only holds true for degrees in music education (since grade-school music teachers are expected to be able to teach lots of instruments at a beginning level). Performance majors are typically only required to play their major instrument (violin, tuba, voice, whatever), plus some sort of basic piano proficiency requirement. Composition majors (such as I was) are typically only required to have the basic piano proficiency (though I do in fact play several other instruments, and I've noticed that many other composers seem to as well). As far as I know, the same is true for conducting majors. (Most conductors I know are violinists or pianists, but this is certainly not required, and one of the best conductors I've worked with plays bass and cello -- and no piano.) --MarnenLaibowKoser

Actually, lots of people would argue that building architects are worse for not actually implementing their designs. Frank Lloyd Wright is the biggest example, since so many of his buildings look great but are absolutely unlivable. If you read ChristopherAlexander, especially TheTimelessWayOfBuilding, you'll see this in detail. Also in HowBuildingsLearn, which has a section specifically devoted to MagazineArchitects: people who design for magazine layouts and not livability.

I'm noticing a severe case of "Title-isis" - or the problem of not knowing what a certain title really means. I myself had a brief bout of it, and have recently settled on "Simplist" (@ http://udidahan.weblogs.us) - or "He who makes things simple" - keeping in vein with one of the central XP tenets. I think it would be interested to allow all people involved in a project to come up with their own titles instead. At the very least you could get valuable insight as to what people think their place in the project really is. *What's wrong with "simplifier"?*'

An interesting question: How often are architects held accountable for the success or failure of their designs? On many projects, the architect reports to the project lead or ProgramManager (according to Brooks, one shouldn't be both architect and manager, except for possibly on very small projects). I've seen quite a few projects where the project was late and the developers and ProgramManager consequently caught hell, but the architect emerged unscathed. (OTOH, when a project is successful, architects - like many others higher up the food chain - are quick to claim credit).

Is it even possible? How would one assign responsibility for project failure to a poor architecture (rather than errors in requirements, poor scheduling/budgeting, or implementation issues)?

ArchitectsDontCode is much less of an AntiPattern if the architect is responsible for the success of their designs. (That alone might prod architects to get more involved in implementation.)

Perhaps Architects Look At the Wrong Thing

Let's step back and look at what is going on. What to software architects do? They create models representing the technology. What is the audience? Developers who are already knowledgeable in the technology. What does the audience really need to know? The operational details of how the product is to be used. The writers of software do not need to be told how to organize the software; they need to know what the software needs to do. Optimizing the organization of the software will not improve the success of the software; making sure the software meets the needs of the user is critical. Developers can do an adequate job of partitioning software and don't really need to have someone tell them how to do it. Developers do need to know how the software is to be used, but unfortunately most developers never get this type of information until well after the software is deployed and the users complain about the "obvious" mistakes the developers made.

After all is said and done, as long as you know who is supposed to do what you will survive. The problem I have seen is the technical leads or senior developers get caught in muddy waters because there is no clear definition of Architect vs Tech Lead. I am in a current situation where I am having to explain UML, design details, etc to an architect who has never written a line of distributed code yet his title is Application Architect. There is a current battle between him and me and nobody wants to define who has what responsibility; needless to say, I am walking.

Been there, done that. If it's a good job, you don't need to be walking. Try to PutIdeasInTheirMouth, it often times works beautifully.

Being an architect, I can tell you, that the architects you speak of, are merely examples of **bad architects** not, **architects that don't code**. I coded COBOL, C, C++, and Java, for many, many years. I've designed many, many OO systems, and sometimes the **pure** OO approach isn't always the right one. This architect's mistake is that he is asking you to dramatically alter your design, after you've already written **working** code. Data access best practices, are well known by this date in time, and if this person doesn't know of them, well, they aren't much of an architect, now, are they? Not coding isn't the problem...not understanding what we''ve learned about J2EE, Java, OO and data access, over the last 10 years, is.

We need to distinguish between internal and external design. The architect's primary responsibility is the external design - those aspects of the product that the user sees and manipulates. This is for all intents and purposes the requirement statement for the developers and should use as many views as are necessary to avoid any misunderstanding about what the product must do

(UserStory, UseCase, InteractionModel, etc.).

TheSourceCodeIsTheDesign applies to the internal design - those aspects of the product hidden from the user except by inference from its external behavior. This assumes you include graphical descriptions of the objects, components and interfaces (e.g. using UML) as an aspect of the code.

Whether the architect also contributes to the internal design or not, he must have development experience so that he knows what's possible, what's easy, what's hard and what's risky. He should also have a feel for the capabilities of the development team (a) so that he doesn't produce a design they lack the skill to build and (b) so that he can prepare a reasonable resource plan for the project.

Many years ago, IBM ruined the title "Systems Analyst" by giving it to older programmers in lieu of a raise. A similar thing is happening with "Architect" which is why a lot of the contributors to this page have a problem distinguishing Architect from Technical Lead and other senior programmer roles.

The JobTitles problem.

Ok, how's this for an argument in favor of ArchitectsDontCode: it separates the high-level design (what) from the implementation (how). For example, let's say I design an application. As a separate decision, we may decide to use Struts or Rails for the implementation. I know Java, I don't really know Ruby. Why should the implementation choice make the design bad? We design applications to isolate the effects of such change - why not design the organization and process with the same philosophy?

```
 From I'm an architect. (I prefer to call myself a "technical
lead".)... above.
```

I think the problem is that typically there are not enough "technical lead"s to address the communication gap between the architect and the programmers. A good "technical lead" should be able to understand the issues of concern to architects, and a good architect should be able to understand the concerns of development and maintenance programmers. A willingness to align and communicate should be the trait of both types of people.

It is too much to ask for an architect to be conversant with the details of programming (e.g. give estimates on project design and programming costs). Likewise, one should not ask the "technical lead" about the implications of infrastructure decisions.

I could be wrong, but the use of the AntiPatternTemplate seemed to be messed up here. The text under "problem" is what should have been under "resulting context", et cetera. I shuffled things around and added some bland

text. --JesseMillikan

Consider this: You want to use some particular functionality from a library. The library is offered in a lib file which you simply link into your project to do the build. The functions you wish to call are provided in a straightforward interface for your chosen language. You link the library in, call the functions, and away you go.

Guess what? The library was written in assembly. Oh, no, wait -- it was written in C++. No, it was Java. No, it was...whatever. It matters not what language or object orientation scheme or development methodology or anydamnthing else was used to create the library. The point is that the library is complete, sufficient, and working.

An architect designs components to work together. As an architect I care not whether a particular component is done in C++ or as an object-oriented subsystem or an off-the-shelf library or even if it is done in hardware. My concern is that the durn thing works as advertised. Therefore, demanding that I do coding so that I can design architecture is specious and wrong-headed.

Leave architects to do their work. If you are concerned about code then deal with coding issues such as optimization and object design after you have something working in the first place. C'mon, folks. This stuff should be fairly obvious to even the most casual observer around here.

-- MartySchrader

From my own experience both as a programmer and most recently as a technical architect - sometimes it does matter how things are implemented, and understanding the implications of those variations is valid when you are integrating all these parts together. Given the complexity of some systems and libraries - can you with a straight face tell me you know without a doubt that using said framework will not cause side effects when utilized in a given way? A good example from my own experiences is a vendor architect who chose a given java library to use for implementing a daemon-like process; when this particular code would run for several months it would actually create a memory leak-like situation in Java (not possible you might say at first blush). Eventually the application would stop working - and people would start complaining. This problem turned out to be a side effect to how the library framework functioned, and the design of the program. Those two things taken together created a situation where memory would be allocated in the command loop - and never released because the library depended on garbage collection to happen on objects created when the loop went out of scope - rather than specifically deallocating the object memory upon completion of the given function. I am sure you can appreciate that in a daemon process, the main command loop will never go out of scope until you either explicitly kill or restart the application, or reboot the machine. If the architect in question had known about this, he could have either chosen a different library, or specified for the programmers to code around the problem. We

ended up retooling the whole effort using Perl - and it ran faster (since we don't have the overhead of loading into memory a heavyweight library) and has been running for 8 years now without issue. What you describe is a problem - not a strength. Architects who understand this are better architects.

Actually, the problem here was one of insufficient specificity. (This dependence on "garbage collection" -- a term I find highly suspect -- is part of the reason I loath the use of Java and such languages in an embedded or deterministic setting.) If the product needed to run as a servlet for long periods without any external maintenance, then that consideration should have affected the choices surrounding memory management and the eventual implementation of memory management in code. Trusting to a non-deterministic subsystem to "take care of" vital matters like clearing memory over the long term is an invitation to disaster. An architect is responsible for finding and quashing these problems long before they ever arise.

See: NonCodingArchitectsSuck, AuthorsDontCode, CommitteesDontCode, ProfessorsDontCode, the theatrical section of DontPlanDesigns, TheArchitectSong

CategoryAntiPattern, CategoryRant

Last edit August 23, 2013, See github about remodling.