What's your most controversial programming opinion?

Asked 15 years, 11 months ago Modified 1 year, 11 months ago

Viewed 312k times

363

votes

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

This is definitely subjective, but I'd like to try to avoid it becoming argumentative. I think it could be an interesting question if people treat it appropriately.

The idea for this question came from the comment thread from my answer to the "What are five things you hate about your favorite language?" question. I contended that classes in C# should be sealed by default - I won't put my reasoning in the question, but I might write a fuller explanation as an answer to this question. I was surprised at the heat of the discussion in the comments (25 comments currently).

So, what contentious opinions do *you* hold? I'd rather avoid the kind of thing which ends up being pretty religious with relatively little basis (e.g. brace placing) but examples might include things like "unit testing isn't actually terribly helpful" or "public fields are okay really". The important thing (to me, anyway) is that you've got reasons behind your opinions.

Please present your opinion and reasoning - I would encourage people to vote for opinions which are well-argued and interesting, whether or not you happen to agree with them.

language-agnostic

Share

edited May 23, 2017 at 12:10

community wiki
6 revs, 4 users 61%
Jon Skeet

Comments disabled on deleted / locked posts / reviews

407 Answers

Sorted by: Highest score (default)

9 votes

Preconditions for arguments to methods/functions should be part of the language rather than programmers checking it always.

Share

answered Jan 3, 2009 at 1:36

community wiki
kal

1 I like it, but it is controversial? – erikkallen Jan 3, 2009 at 22:53

9
votes

Requirements analysis, specification, design, and documentation will almost never fit into a "template." You are 100% of the time better off by starting with a blank document and beginning to type with a view of "I will explain this in such a way that if I were dead and someone else read this document, they would know everything that I know and see and understand now" and then organizing from there, letting section headings and such develop naturally and fit the task you are specifying, rather than being constrained to some business or school's idea of what your document should look like. If you have to do a diagram, rather than using somebody's formal and incomprehensible system, you're often better off just drawing a diagram that makes sense, with a clear legend, which actually specifies the system you are trying to specify and communicates the information that the developer on the other end (often you, after a few years) needs to receive.

[If you have to, once you've written the real documentation, you can often shoehorn it into whatever template straightjacket your organization is imposing on you. You'll probably find yourself having to add section headings and duplicate material, though.]

The only time templates for these kinds of documents make sense is when you have a large number of tasks which are very similar in nature, differing only in details. "Write a program to allow single-use remote login access through this modem bank, driving the terminal connection nexus with C-Kermit," "Produce a historical trend and forecast report for capacity usage," "Use this library to give all reports the ability to be faxed," "Fix this code for the year 2000 problem," and "Add database triggers to this table to populate a software product provided for us by a third-party vendor" can **not** all be described by the same template, no matter what people may think. And for the record, the requirements and design diagramming techniques that my college classes attempted to teach me and my classmates could not be used to specify a simple calculator program (and everyone knew it).

Share

answered Jan 3, 2009 at 10:46

community wiki
skiphoppy

9
votes

For a good programmer language is not a problem.

It may be not very controvertial but I hear a lot o whining from other programmers like "why don't they all use delphi?", "C# sucks", "i would change company if they forced me to use java" and so on.
What i think is that a good programmer is flexible and is

able to write good programms in any programming language that he might have to learn in his life

Share

answered Jan 5, 2009 at 11:20

community wiki
agnieszka

On the other hand, I *would* change company if, say, I was told that the rest of my job (forever) would be in GWBasic. There's a significant difference in how easy it is to express designs in different languages. – Jon Skeet Jan 6, 2009 at 8:55

yeah, of course it's not applicable to all situations. but still a programmer has to be flexible to some extent because this is what computer science is all about - constant change. – agnieszka Jan 7, 2009 at 12:19

Totally agreed. I hate those religious language wars :/ – driAn Jan 7, 2009 at 14:31

While I agree that a good programmer can understand any language, working with it 40+ hours a week is a different story. I can understand VB.NET just fine, but I don't want to spend most of my day plowing through it! – Cameron MacFarland Jan 8, 2009 at 4:35

2 I can agree with this. The real truth here is that there is a tool for every job. Sometimes that tool may be Perl. Sometimes it may be vbScript, sometimes Java, sometimes C#, and sometime even C++... The good developer knows WHICH tool is right for the job. – LarryF Jan 14, 2009 at 23:57

Sometimes jumping on the bandwagon is ok

9
votes

I get tired of people exhibiting "grandpa syndrome" ("You kids and your newfangled Test Driven Development. *Every* big technology that's come out in the last decade has sucked. Back in my day, we wrote *real* code!"... you get the idea).

Sometimes things that are popular are popular for a reason.

Share

answered Jan 10, 2009 at 0:25

community wiki
Jason Baker

3 Not controversial enough. To be controversial, replace sometimes with always. – Coding With Style Jul 4, 2009 at 22:11

My problem is that otherwise good ideas become bandwagons. My favorite example is OOP, a useful idea that became a binge. In most of the performance tuning I do, the culprit, ultimately, is that a Queen Mary was built, when a rowboat would have sufficed. – Mike Dunlavey Feb 6, 2010 at 16:14

@Mike Dunlavey - I agree 100%. But it's not fair to reject an idea on that basis (which a lot of people do). – Jason Baker Feb 6, 2010 at 16:34

... talk about old-time code, how about this: `//SYSUT2 DD UNIT=(TAPE1600,,DEFER),VOL=SER=SPROOOF,LABEL=(1,NL),DISP=(,KEEP)` cranked out standing up at a keypunch. – Mike Dunlavey Feb 6, 2010 at 17:48

9
votes

VB 6 could be used for good as well as evil. It was a Rapid Application Development environment in a time of over complicated coding.

I have hated VB vehemently in the past, and still mock VB.NET (probably in jest) as a Fisher Price language due to my dislike of classical VB, but in its day, nothing could beat it for getting the job done.

Share

answered Jan 18, 2009 at 10:39

community wiki
johnc

9
votes

Code Generation is bad

I hate languages that require you to make use of code generation (or copy&paste) for simple things, like JavaBeans with all their Getters and Setters.

C#'s AutoProperties are a step in the right direction, but for nice DTOs with Fields, Properties and Constructor parameters you still need a lot of redundancy.

Share

answered Jan 25, 2009 at 18:03

community wiki
Lemmy

3 Code Generation is bad... so do you hate compilers also? (Hint: code generation is a broad subject, don't be deceived by crappy languages/frameworks). – MaD70 Nov 5, 2009 at 23:45

9
votes

MIcrosoft is not as bad as many say they are.

Share

answered Aug 1, 2009 at 8:25

community wiki
Aftershock

9
votes

If you need to read the manual, the software isn't good enough.

Plain and simple :-)

Share

edited Oct 31, 2009 at 17:44

community wiki
2 revs, 2 users 80%
simonjpascoe

1 I agree that there is a lot of software that could do without a manual if it had been designed with a greater emphasis on usability. But even when you can figure out stuff without a manual, having a manual might let you figure out stuff quicker! – Captain Sensible Jan 26, 2009 at 14:15

I kinda agree with this. I've seen quite a few applications that were badly designed but sometimes the bad design is by the customers own request (this is how our previous application worked and we don't want to change to much even if it sucks). In these circumstances, whether something is good enough or not is not decided by the development team but by the customer. You may disagree but the customer is always right ;-) – Captain Sensible Apr 22, 2010 at 8:31

9 votes

Programming is neither art nor science. It is an engineering discipline.

It's not art: programming requires creativity for sure. That doesn't make it art. Code is designed and written to work properly, not to be emotionally moving. Except for whitespace, changing code for aesthetic reasons breaks your code. While code can be beautiful, art is not the primary purpose.

It's not science: science and technology are inseparable, but programming is in the technology category. Programming is not systematic study and observation; it is design and implementation.

It's an engineering discipline: programmers design and build things. Good programmers design for function. They understand the trade-offs of different implementation options and choose the one that suits the problem they are solving.

I'm sure there are those out there who would love to parse words, stretching the definitions of art and science to include programming or constraining engineering to mechanical machines or hardware only. Check the dictionary. Also "The Art of Computer Programming" is a different usage of art that means a skill or craft, as in "the art of conversation." The product of programming is not art.

Share

answered Nov 13, 2009 at 1:12

community wiki
Paul

8
votes

We do a lot of development here using a Model-View-Controller framework we built. I'm often telling my developers that we need to violate the rules of the MVC design pattern to make the site run faster. This is a hard sell for developers, who are usually unwilling to sacrifice well-designed code for anything. But performance is our top priority in building web applications, so sometimes we have to make concessions in the framework.

For example, the view layer should never talk directly to the database, right? But if you are generating large reports, the app will use a lot of memory to pass that data up through the model and controller layers. If you have a database that supports cursors, it can make the app a lot faster to hit the database directly from the view layer.

Performance trumps development standards, that's my controversial view.

Share

answered Jan 2, 2009 at 23:02

community wiki
jjriv

An excellent example of how sometimes rules are made to be broken. Do everything right but be prepared to do some things wrong from necessity. – Kendall Helmstetter Gelner Jan 5, 2009 at 5:43

---

1 Performance trumps development standards -- if it is too poor to stand. As long as performance is not a problem, there is no need to fix it. – Aaron Digulla Feb 27, 2009 at 14:53

Don't forget, what is considered "right" in terms of development standards was just somebody's common-sense temporary opinion that happened to get picked up by a lot of people. It is not a commandment from "on high" - common sense can change but is always useful. Good work. – Mike Dunlavey Feb 6, 2010 at 15:56

8
votes

I believe the use of try/catch exception handling is worse than the use of simple return codes and associated common messaging structures to ferry useful error messages.

Littering code with try/catch blocks is not a solution.

Just passing exceptions up the stack hoping whats above you will do the right thing or generate an informative error is not a solution.

Thinking you have any chance of systematically verifying the proper exception handlers are avaliable to address anything that could go wrong in either transparent or opague objects is not realistic. (Think also in terms of late bindings/external libraries and unecessary dependancies between unrelated functions in a call stack as system evolves)

Use of return codes are simple, can be easily systematically verified for coverage and if handled properly forces developers to generate useful error messages rather than the all-too-common stack dumps and obscure I/O exceptions that are "exceptionally" meaningless to even the most clueful of end users.

--

My final objection is the use of garbage collected languages. Don't get me wrong.. I love them in some circumstances but in general for server/MC systems they have no place in my view.

GC is not infallable - even extremely well designed GC algorithms can hang on to objects too long or even forever based on non-obvious circular refrences in their dependancy graphs.

Non-GC systems following a few simple patterns and use of memory accounting tools don't have this problem but do require more work in design and test upfront than GC environments. The tradeoff here is that memory leaks are extremely easy to spot during testing in Non-GC while finding GC related problem conditions is a much more difficult proposition.

Memory is cheap but what happens when you leak expensive objects such as transaction handles, synchronization objects, socket connections...etc. In my environment the very thought that you can just sit back and let the language worry about this for you is unthinkable without significant fundental changes in software description.

Share

answered Jan 2, 2009 at 23:05

community wiki
Einstein

Return codes have the problem of coupling too many elements of a chain of calls to understand what they mean. That is to say, that everything between a called function and something that might handle an error has to understand the return codes, at least to pass them along - that can be a mess.
– Kendall Helmstetter Gelner Jan 5, 2009 at 5:45

---

1 My general advice is to follow a convention and don't fall into the trap of attempting to have them indiciate specific error conditions. At each level you should take steps to ensure

meaning is normalized. (Which ususally isn't hard/necessary if you follow a convention) – Einstein Jan 5, 2009 at 6:33

Good error code compared with bad exception code is better. But then again, there is good exception handling code, where exceptions are thrown and caught only where it makes sense... good exception code separates error handling from the error, and need not be replicated in each function of the stack – David Rodríguez - dribeas Jan 5, 2009 at 18:23

If a GC platform is not right for your particular situation, use good judgmenet and don't use it. It's as simple as that. – Captain Sensible Apr 22, 2010 at 8:39

8 votes

The best programmers trace all their code in the debugger and test all paths.

Well... the OP said controversial!

Share

answered Jan 3, 2009 at 3:52

community wiki
Enigme

Please justify your position. Note: test all paths requires that you only write paths you can test. Mindless error handlers go away. – Jay Bazuzi Jan 3, 2009 at 17:41

Ever heard of unit tests? Using unit tests you don't need to "test all paths" after each change you made to the code. (Anyway, I think it's is impossible to test all paths except in a tiny little application) – Stefan Steinegger Nov 16, 2009 at 11:41

8
votes

Web services absolutely suck, and are not the way of the future. They are ridiculously inefficient and they don't guarantee ordered delivery. Web services should NEVER be used within a system where both client and server are being written. They are mostly useful for micky mouse mash-up type applications. They should definitely not be used for any kind of connection-oriented communication.

This stance has gotten myself and colleagues into some very heated discussions, since web services is such a buzzy topic. Any project that mandates the use of web services is doomed because it is clearly already having ridiculous demands pushed down from management.

Share

answered Jan 3, 2009 at 6:26

community wiki
Jesse Pepper

My company writes auto-insurance software, and we rely on several off-site web services to verify VIN numbers and run OFAC checks on people. We also make some of our APIs available through web services to third-party vendors. How would you suggest our software be written without web services? – Juliet Jan 4, 2009 at 3:55

@Juliet: what in " Web services should NEVER be used within a system **where both client and server are being written** " do you not understand? It's clear that in your situation you don't control both parts of the system, so your rhetorical question is irrelevant. – MaD70 Nov 6, 2009 at 1:21

8

votes

Relational databases are awful for web applications.

For example:

- threaded comments
- tag clouds
- user search
- maintaining record view counts
- providing undo / revision tracking
- multi-step wizards

Share

edited Jan 9, 2009 at 17:07

community wiki
2 revs, 2 users 95%
Mike Judge

1 The reason OODB didn't take off for web apps is because web apps are the single area where scalability and speed matter most - and OODB fall flat when load gets high. That's why MySQL took off instead of something more robust like Postgres, because of sheer read speed and scalability. – Kendall Helmstetter Gelner Jan 5, 2009 at 5:31

1 kendall, that's just trash. the biggest databases in the world have traditionally been oodbs. they handle all kinds of workload. – nes1983 Apr 12, 2009 at 10:02

Only deep ignorance can prevent someone to implement such things even in SQL, which is a badly designed language and not faithful to relational data model. – MaD70 Nov 6, 2009 at 0:28

8 votes

Assembly is the best first programming language.

Share

answered Jan 14, 2009 at 2:59

community wiki
Darvon

+1 for that... probably too hard for most people to grasp... nothing like weeding out the weak ones. ;) – oz10 Jan 14, 2009 at 3:14

I learned Mostek 6502 assembly at 12 and was my second programming language (the first was an unstructured Basic - pure crap). It was easy with a book and some computer magazines (those with long listing of source code). K&R C disgust me even today. – MaD70 Nov 6, 2009 at 1:09

8 votes

The code *is* the design

Share

edited Jan 15, 2009 at 10:58

community wiki
2 revs
bjnortier

8

votes

A good developer needs to know more than just how to code

Share

answered Jan 16, 2009 at 15:03

community wiki
Wiren

3 It's not that I don't agree but I like to see more explanation or examples. – tuinstoel Jan 22, 2009 at 18:12

8

votes

Developers overuse databases

All too often, developers store data in a DBMS that should be in code or in file(s). I've seen a one-column-one-row table that stored the 'system password' (separate from the user table.) I've seen constants stored in databases. I've seen databases that would make a grown coder cry.

There is some sort of mystical awe that the offending coders have of the DBMS--the database can do anything, but they don't know how it works. DBAs practice a black art. It also allows responsibility transference: *"The database is too slow," "The database did it"* and other excuses are common.

Left unchecked, these coders go on develop databases-within-databases, systems-within-systems. (There is a name to this anti-pattern, but I forget what it is.)

Share

answered Mar 30, 2009 at 12:13

community wiki
Stu Thompson

Guessing you are talking about EAV (Entity Attribute Value) database design, which has been the bane of my life for about a year now :) – spooner Jan 10, 2010 at 21:59

8 votes

Sometimes you have to denormalize your databases.

An opinion that doesn't go well with most programmers but you have to sacrifice things like noramlization for performance sometimes.

Share

answered May 10, 2009 at 6:50

community wiki
Artem Russakovskii

Hardware is cheap - logic costs a fortune. – Steven Evers Jun 19, 2009 at 18:18

Corollary: most time, performance suffers with less than 5NF. – just somebody Dec 15, 2009 at 2:23

8

Test Constantly

votes

You have to write tests, and you have to write them FIRST. Writing tests changes the way you write your code. It makes you think about what you want it to actually do before you just jump in and write something that does everything except what you want it to do.

It also gives you goals. Watching your tests go green gives you that little extra bump of confidence that you're getting something accomplished.

It also gives you a basis for writing tests for your edge cases. Since you wrote the code against tests to begin with, you probably have some hooks in your code to test with.

There is not excuse not to test your code. If you don't you're just lazy. I also think you should test first, as the benefits outweigh the extra time it takes to code this way.

Share

edited May 15, 2009 at 14:04

community wiki
2 revs, 2 users 94%
PJ Davis

OMG how did anyone down vote this. Amazing, i'd + 1000 if i could – user34537 Jan 4, 2009 at 7:54

1 Sometimes, watching all your test go green gives you a FALSE confidence, while your code fails somewhere your test didn't anticipate. – Cameron MacFarland Jan 4, 2009 at 8:29

@acidzombie24, you should vote for it if you think it is controversial, not when you agree it. – tuinstoel Jan 4, 2009 at 11:45

@Cameron MacFarland there is no excuse for not doing user testing. The point of the test isn't to cover every edge case from the beginning, it's to make sure your code meets the requirements for what it's supposed to do. No matter how much you test, you'll never cover everything that could happen. – PJ Davis Jan 6, 2009 at 0:26

2 You're accruing "offensive" votes... suggest you remove the profanity. – Marc Gravell May 15, 2009 at 13:59

8 votes

Premature optimization is **NOT** the root of all evil! Lack of proper planning is the root of all evil.

Remember the old naval saw

> Proper Planning Prevents P*ss Poor Performance!

Share

edited Jul 20, 2009 at 16:15

community wiki
2 revs
WolfmanDragon

8 votes

Correct every defect when it's discovered. Not just "severity 1" defects; **all** defects.

Establish a deployment mechanism that makes application updates immediately available to users, but allows them to choose when to accept these updates. Establish a direct communication mechanism with users that enables them to report defects, relate their experience with updates, and suggest improvements.

With aggressive testing, many defects can be discovered during the iteration in which they are created; immediately correcting them reduces developer interrupts, a significant contributor to defect creation. Immediately correcting defects reported by users forges a constructive community, replacing product quality with product improvement as the main topic of conversation. Implementing user-suggested improvements that are consistent with your vision and strategy produces community of enthusiastic evangelists.

Share

edited Jul 29, 2009 at 3:00

community wiki
2 revs, 2 users 96%
Dave

not really "controversial" - it's the standing practice everywhere I've worked – warren Oct 22, 2009 at 4:22

8
votes

Programming: It's a fun *job*.

I seem to see two generalized groups of developers. Those that don't love it but they are competent and the money is good. The other group that love it to a point that is kinda creepy. It seems to be their life.

I just think it well paying job that is usually interesting and fun. There is all kinds of room to learn something new every minute of every day. I can't think of another job I would prefer. But it is still a job. Compromises will be made and the stuff you produce will not always be as good as it could be.

Given my druthers would be on a beach drinking beer or playing with my kids.

Share

answered Oct 29, 2009 at 18:48

community wiki
ElGringoGrande

8
votes

Assembler is not dead

In my job (copy protection systems) assembler programming is essential, I was working with many hll copy protection systems and only assembler gives you the real power to utilize all the possibilities hidden in the code (like code mutation, low level stuff).

Also many code optimizations are possible only with an assembler programming, look at the sources of any video codecs, sources are written in assembler and optimized to use MMX/SSE/SSE2 opcodes whatever, many game engines uses assembler optimized routines, even Windows kernel has SSE optimized routines:

```
NTDLL.RtlMoveMemory

.text:7C902CD8                push    ebp
.text:7C902CD9                mov     ebp, esp
.text:7C902CDB                push    esi
.text:7C902CDC                push    edi
.text:7C902CDD                push    ebx
.text:7C902CDE                mov     esi,
[ebp+0Ch]
.text:7C902CE1                mov     edi, [ebp+8]
.text:7C902CE4                mov     ecx,
[ebp+10h]
.text:7C902CE7                mov     eax, [esi]
.text:7C902CE9                cld
.text:7C902CEA                mov     edx, ecx
.text:7C902CEC                and     ecx, 3Fh
.text:7C902CEF                shr     edx, 6
.text:7C902CF2                jz      loc_7C902EF2
.text:7C902CF8                dec     edx
.text:7C902CF9                jz      loc_7C902E77
.text:7C902CFF                prefetchnta byte ptr
[esi-80h]
.text:7C902D03                dec     edx
.text:7C902D04                jz      loc_7C902E03
.text:7C902D0A                prefetchnta byte ptr
[esi-40h]
.text:7C902D0E                dec     edx
.text:7C902D0F                jz      short
loc_7C902D8F
.text:7C902D11
.text:7C902D11 loc_7C902D11:
; CODE XREF: .text:7C902D8Dj
.text:7C902D11                prefetchnta byte ptr
[esi+100h]
.text:7C902D18                mov     eax, [esi]
.text:7C902D1A                mov     ebx, [esi+4]
.text:7C902D1D                movnti  [edi], eax
```

```
text:7C902D20                    movnti  [edi+4], ebx
```

So if you hear next time that assembler is dead, think about the last movie you have watched or the game you've played (and its copy protection heh).

Share

edited Nov 12, 2009 at 12:24

community wiki
2 revs, 2 users 98%
user205036

6 Assembler is also essential for breaking copy protections :D – Peter Nov 19, 2009 at 21:03

Wait, copy protection, so you don't like my iTunes Library, do you? – user142019 Dec 20, 2010 at 14:47

8
votes

Writing it yourself can be a valid option.

In my experience there seems to be too much enthusiasm when it comes to using 3rd party code to solve a problem. The option of solving the problem by themselves does usually not cross peoples minds. Although don't get me wrong, I am not propagating to never ever use libraries. What I am saying is: among the possible frameworks and modules you are considering to use, add the option of implementing the solution yourself.

But why would you code your own version?

- Don't reinvent the wheel. But, if you only need a piece of wood, do you really need a whole cart wheel? In other words, do you really need openCV to flip an image along an axis?
- Compromise. You usually have to make compromises concerning your design, in order to be able to use a specific library. Is the amount of changes you have to incorporate worth the functionality you will receive?
- Learning. You have to learn to use these new frameworks and modules. How long will it take you? Is it worth your while? Will it take longer to learn than to implement?
- Cost. Not everything is for free. Although, this includes your time. Consider how much time this software you are about to use will save you and if it is worth it's price? (Also remember that you have to invest time to learn it)
- You are a programmer, not ... a person who just clicks things together (sorry, couldn't think of anything witty).

The last point is debatable.

Share

answered Dec 3, 2009 at 13:38

community wiki
Stefan Schmidt

8
votes

Relational database systems *will* be the best thing since sliced bread...

... when we (hopefully) get them, that is. SQL databases suck so hard it's not funny.

What I find amusing (if sad) is *certified DBAs* who think an SQL database system is a relational one. Speaks volumes for the quality of said certification.

Confused? Read C. J. Date's books.

edit

Why is it called *Relational* and what does that word mean?

These days, a programmer (or a certified DBA, *wink*) with a strong (heck, *any*) mathematical background is an exception rather than the common case (I'm an instance of the common case as well). SQL with its *tables*, *columns* and *rows*, as well as the joke called Entity/Relationship Modelling just add insult to the injury. No wonder the misconception that *Relational* Database Systems are called that because of some *Relationships* (Foreign Keys?) between *Entities* (tables) is so pervasive.

In fact, *Relational* derives from the *mathematical* concept of relations, and as such is intimately related to set theory and functions (in the mathematical, not any programming, sense).

[http://en.wikipedia.org/wiki/Finitary_relation][2]:

> In mathematics (more specifically, in set theory and logic), a **relation** is a property that assigns truth values to combinations (*k*-tuples) of *k* individuals. Typically, the property describes a possible connection between the components of a *k*-tuple. For a given set of *k*-tuples, a truth value is assigned to each *k*-tuple according to whether the property does or does not hold.
>
> An example of a ternary relation (i.e., between three individuals) is: "*X* was-introduced-to *Y* by *Z*", where (X,Y,Z) is a 3-tuple of persons; for example, "Beatrice Wood was introduced to Henri-Pierre Roché by Marcel Duchamp" is true, while "Karl Marx was introduced to Friedrich Engels by Queen Victoria" is false.

Wikipedia makes it perfectly clear: in a SQL DBMS, such a ternary relation would be a "*table*", not a "*foreign key*" (I'm taking the liberty to rename the "columns" of the relation: X = who, Y = to, Z = by):

```
CREATE TABLE introduction (
  who INDIVIDUAL NOT NULL
, to INDIVIDUAL NOT NULL
, by INDIVIDUAL NOT NULL
, PRIMARY KEY (who, to, by)
);
```

Also, it would contain (among others, possibly), this "*row*":

```
INSERT INTO introduction (
  who
, to
, by
) VALUES (
  'Beatrice Wood'
, 'Henri-Pierre Roché'
, 'Marcel Duchamp'
);
```

but not this one:

```
INSERT INTO introduction (
  who
, to
, by
) VALUES (
  'Karl Marx'
, 'Friedrich Engels'
, 'Queen Victoria'
);
```

Relational Database Dictionary:

> **relation (mathematics)** Given sets *s1*, *s2*, ..., *sn*, not necessarily distinct, *r* is a relation on those sets if and only if it's a set of *n*-tuples each of which has its first element from *s1*, its second element from *s2*, and so on. (Equivalently, *r* is a subset of the Cartesian product *s1* x *s2* x ... x *sn*.)
>
> Set *si* is the *i*th domain of *r* (*i* = 1, ..., *n*). *Note*: There are several important logical differences between relations in mathematics and their

> relational model counterparts. Here are some of them:
>
> - Mathematical relations have a left-to-right ordering to their attributes.
> - Actually, mathematical relations have, at best, only a very rudimentary concept of attributes anyway. Certainly their attributes aren't named, other than by their ordinal position.
> - As a consequence, mathematical relations don't really have either a heading or a type in the relational model sense.
> - Mathematical relations are usually either binary or, just occasionally, unary. By contrast, relations in the relational model are of degree *n*, where *n* can be any nonnegative integer.
> - Relational operators such as JOIN, EXTEND, and the rest were first defined in the context of the relational model specifically; the mathematical theory of relations includes few such operators.
>
> And so on (the foregoing isn't meant to be an exhaustive list).

Share

edited Dec 16, 2009 at 1:53

community wiki

2 revs
just somebody

Would you agree that today's RDMSs *support* the relational model however rarely are the schema designers implementing it? – Jé Queue Dec 15, 2009 at 5:38

You're asking a loaded question, but do you consider DB2 &| Oracle systems that don't support a true *relation*al model? – Jé Queue Dec 15, 2009 at 23:09

yes. SQL database systems are just that: SQL database systems, not relational database systems. – just somebody Dec 15, 2009 at 23:43

Do you mean Object Databases when you say relational databases? That is, db4o et al.? Relational Database system in my opinion are systems where you model relations between entities, also known as Foreign Keys and Update/Delete Cascades. Sadly, most of the time these entities are flat 2-Dimensional tables in RDBMS... – Michael Stum Dec 16, 2009 at 0:11

@Michael Stum: no, see expanded answer, and excuse me if it's not very coherent. It's well past midnight here and I'm almost done with second bottle of wine. – just somebody Dec 16, 2009 at 1:55

7 votes

[There is no such thing as Object-Oriented programming.]

Share

edited Jan 2, 2009 at 15:04

community wiki

2 revs
Apocalisp

The problem I have with that article is that it argues that OOP doesn't model the real world properly and so it doesn't exist. I agree that OOP is a poor real-world model but that doesn't mean it doesn't exist. – Cameron MacFarland Jan 2, 2009 at 14:48

@Cameron MacFarland: That's not what the article argues at all. It argues that there's no distinction between "OOP" and other kinds of programming, other than a rhetorical one. – Apocalisp Jan 2, 2009 at 15:03

Why is there no reference to ADT which I believe OOP was sprung from? – epatel Jan 2, 2009 at 19:03

1 @Apocalisp: You're right, I only skimmed the article. Now that I've read it properly, he compared making distinctions between code styles with making distinctions about race by using the argument made by capitalist libertarians, who believe in things that lead to slavery and killing poor people. – Cameron MacFarland Jan 3, 2009 at 6:56

1 See I told you it was controversial. Enough to draw an ad hominem with a non-sequitur and a straw man in a single sentence. I'm impressed. – Apocalisp Jan 3, 2009 at 20:07

7 votes

Write your spec when you are finished coding. (if at all)

In many projects I have been involved in, a great deal of effort was spent at the outset writing a "spec" in Microsoft Word. This process culminated in a "sign off" meeting when the big shots bought in on the project, and after that meeting nobody ever looked at this document again. These documents are a complete waste of time and don't reflect how software is actually designed. This is not to say there are not other valuable artifacts of application design. They are usually contained on index cards, snapshots of whiteboards, cocktail napkins and other similar media that provide a kind of timeline for the app design. These are usually are the real specs of the app. If you are going to write a Word document, (and I am not particularly saying you should) do it at the end of the project. At least it will accurately represent what has been done in the code and might help someone down the road like the the QA team or the next version developers.

Share

answered Jan 16, 2009 at 15:23

community wiki
Andrew Cowenhoven

"nobody ever looked at this document again" Not true. When I started at a new job I was given "the spec" folder, and told to read it as my first task. Then I started asking "where's this feature" the answer was "I didn't know that was in the spec." The spec folder was given to all new employees.
– Cameron MacFarland Jan 22, 2009 at 22:36

Yes, that happened to me too - several times. I stand corrected.
– Andrew Cowenhoven Jan 30, 2009 at 14:13

I think this is *highly* situational. Do you need a spec for an internal project that you'll write in 3 days? Probably not. Are you willing to bet a multi-million dollar project on the client understanding everything you say to the letter? I would hope not. – Jason Baker Feb 6, 2010 at 16:38

7
votes

That best practices are a hazard because they ask us to substitute slogans for thinking.

Share

answered Jan 25, 2009 at 14:08

community wiki
Flinkman

The Blind following the blind. – WolfmanDragon May 23, 2009 at 18:12

7
votes

Social skills matter more than technical skills

Agreable but average programmers with good social skills will have a more successful carreer than outstanding programmers who are disagreable people.

Share

edited Jan 27, 2009 at 9:45

community wiki
2 revs
Diego Deberdt

+1 I couldn't agree more. Building software is a social activity more than a technical one. – JuanZe Oct 13, 2009 at 21:34