

# The Programming Profession

from the

## Programmers Guild

### The Guild Wins Legal Settlement in Discrimination Case

The Programmers Guild and the United States Department of Justice won a legal settlement from Adea Group of Dallas TX.

Adea had launched a well-publicized campaign targeted at hiring unemployed H-1B workers who had remained in the U.S. However, such hiring, based upon immigration status, is illegal.

The Guild filed a complaint with the Office of Special Council in the Justice Department regarding these hiring practices. All parties agreed that these actions were not legal and came to a settlement agreement.

Adea has agreed not to engage in hiring based upon immigration status. It has also agree to train its employees it to the law.

### *Hiring Based on Immigration Status Is Illegal*

If you see job ads asking for H-1B workers here's what you should do:

1. Make a hard copy of the ad. Note where you found the ad and on which day.
2. See if you can find other ads from the same company asking for H-1B workers.
3. Send a resume to the company and get your friends and coworkers to send a resume as well.
3. Contact the guild at [guild@colosseumbuilders.com](mailto:guild@colosseumbuilders.com). We will file a complaint on your behalf.

### Americans Replaced by H-1B Workers.

Exult Corporation is in the process of laying off about 70 programmers and "outsourcing" to an H-1B bodyshop called Hexaware. Programmers were notified in December 2001 that they had to train their replacements to collect severance.

Exult is a new company that specializes in the "outsourcing" of Human Resources functions. Most of those programmers losing their jobs were transferred to Exult from other companies, notably Bank of America and BP, when those companies engaged Exult.

Don't expect to find this story in the mainstream press.

### DoL Calls H-1B Training Grants Useless - We Told you So

According to the DoL Budget "The H-1B Training Grant Program is supposed to train U.S. workers for jobs in which labor shortages have caused employers to hire foreign workers through the H-1B visa program. These highly educated workers typically work in the high-tech and health care industries. Unfortunately, DOL's \$138 million H-1B Training program, which is financed through a \$1,000 fee paid by employers, never has filled and has no prospect of filling these labor shortages. At times, funds wind up training workers for decidedly low-tech jobs. One grant financed training for cable installers; another trained licensed practical nurses; while a third was

open only to union members in the entertainment industry. The budget will take the program's H-1B fees funding and redirect it to eliminate large backlogs in the permanent alien labor program, thereby better serving workers and employers alike."

### San Diego Programmer a Big Winner

Ron de Frates of San Diego CA is the lucky winner of the Guild's membership contest. Ron won a copy of professional editions of:

- o Delphi
- o C++Builder
- o Kylix (Delphi for Linux)

Courtesy of Borland.

The Guild expresses its appreciation to Borland for supporting our recruitment efforts by providing us with copies of the best development tools on the market.

THANK YOU BORLAND!!!!

### AAA Eliminating AEA Discount

The Southern California Automobile club is terminating its American Engineering Association discount effective March 1st. Guild members who are in the AEA will wish to contact their local AAA office about this change.

### Comments

Letters to the Editor and submissions for publication may be made to: [newsletter@programmersguild.com](mailto:newsletter@programmersguild.com)

or

The Programmers Guild  
PO Box 1250  
Summit NJ 07902-1250

## Chairman's Message

*By John Miano*

If you have had this newsletter delivered to your home, I would like to welcome you as a charter member of the Programmers Guild. You are the pioneers of the programming profession, those who are willing to invest in building its advancement.

The Programmers Guild has come a long way since the idea was proposed in 1998. Since then nearly 1,500 programmers decided to join us. Making the transition last year to a dues-supported organization was the biggest step leap so far in the organization's development. This newsletter is a tangible result of this transition.

Many people have asked me "What's the point of having a printed newsletter when we could use e-mail?" The answer to this question is quite simple: A printed newsletter allows the Guild to communicate directly with your home. E-mail addresses change frequently and leave no forwarding address.

Major Guild activities taking place right now include:

Certification

Ways to improve the process of software development

Legislative Issues

Most of the action takes place at local meetings.

We still have a long way to go as an organization. Right now, only a tiny fraction of the programming profession is with us. We need to build up the membership rolls in order to offer many of the benefits, such as insurance, that you have asked for. You have to start somewhere and we have made that start.

## Local Programmers Guild Activities

### New Jersey

The New Jersey Chapter of the Programmers Guild has been meeting on the 2nd Tuesday of each month, usually at the Marco Polo Restaurant in Summit.

In September, the NJ Chapter conducted a joint meeting with the U.S. Department of Commerce as part of their study on training needs for the programming profession.

Assistant Secretary Bruce Mehlman thanked the Guild for its participation.

For more information on how to get involved with the Programmers Guild's NJ Chapter, contact John Miano at [guild@colosseumbuilders.com](mailto:guild@colosseumbuilders.com).

### Charlotte, NC

A Programmers Guild chapter is being organized in Charlotte NC.

Contact Bob Evans at [bobevans19@yahoo.com](mailto:bobevans19@yahoo.com) if you are interested.

### Cleveland, OH

The Cleveland Guild Chapter has started meeting regularly to discuss the state of the programming profession. It has created the Yahoo group [pgohio@yahogroups.com](mailto:pgohio@yahogroups.com).

Contact Paul Hanrahan at [hanrahan\\_paul@ameritech.net](mailto:hanrahan_paul@ameritech.net) for more information.

### Minneapolis, MN

Over the past year, the MN Chapter has had several monthly meetings and has setup a yahoo group called PGMinn to keep everyone in the group informed of current news articles. There are about 40 members in the MN group.

Several members attended town meetings for Representative Jim Ramstad and spoke with him about

our concern with the H-1B visa program. Members attended Senator Paul Wellstone's campaign kickoff celebration and spoke with him about the inability of Americans to find IT jobs. Five unemployed members met with one of his aides to discuss our current job situations. We gave him a copy of the Star Tribune article that Sherri Cruz wrote about our job situations and her article on large corporations in the Twin Cities that are outsourcing their IT support to Indian-owned consulting companies. We are writing letters to the Department of Labor explaining our current job situations and the abuse of the H-1B visa laws. Wellstone's office will send the letters to the Department of Labor and request an answer from them. They have also setup another meeting with us and are inviting a representative from unemployment, dislocated worker funds, and the Department of Labor. We are asking for enforcement of the current law - that H-1B workers be paid the prevailing wage and that an H-1B worker can only be hired if a qualified American cannot be found for the job.

The Minnesota chapter maintains the Yahoo group PGMinn. For information Minneapolis organizing, contact Linda Nesheim at [Goldstar01@aol.com](mailto:Goldstar01@aol.com)

### No Group in Your Area

Start one! All it takes is a reservation at a local watering hole and invitations to your friends and coworkers. The Guild can help you out by announcing your meeting to people we have contact with in your area.

## In defense of offshore programmers

By Edward Nilges

*I pity the poor immigrant... - Bob*

*Dylan*

The biggest mistake labor, organized, unorganized, or disorganized, can make is to associate itself in any way with racism and hatred of "foreigners", yet this is the mistake being made in some calls to end the H-1B program by means of which corporations here import cheap but highly skilled programmers to replace American programmers.

Many people who oppose H-1B are honestly troubled by the exploitation of the offshore programmer who is often hard working and docile for a very good reason, as well as by the threat to American jobs. When an American programmer is terminated he has only to turn to the Monster board or Dice.COM for another job starting Monday morning. For a programmer in the H-1B program, termination means a bogus journey, not an excellent adventure. He has to go back to his home country and reapply through H-1B. H-1B exploits offshore programmers because they can't talk back and walk when management decides on some idiotic policy such as a ban on the development of any new objects in a C++ shop or mandatory unpaid overtime. H-1B is used by management to obtain slaves.

But it appears to me (from Internet exchanges on the topic) that at the grassroots dislike of the H-1B program is not pity or sympathy for, or solidarity with, the poor immigrant. It's good old fashioned American xenophobia, hatred of the foreigner, coupled with anti-intellectualism in programming.

Americans know that it's contradictory to hate foreigners for with the

exception of the Native Americans, we're all foreigners here. Nonetheless, American workers have continually confused their real needs with easy-to-organize movements against "those people." I'm partly Irish, and in the 19th century, my people were those people: according to the Know-Nothings of the 1830s (small Protestant tradespeople and skilled craftsmen) we were drunk most of the time, priest-ridden and we talked too much. Nowadays "those people" are H-1Bs and second and third generation Chinese Americans who according to the xenophobic text entirely too sober, of no particular religion, and way to good at getting into college. And of course, right next door to the xenophobia of "those people" is racism which takes the talk to the max, and which paints ugly pictures of "those people" who are way to good at basketball.

Now, many people say they object to the H-1B program in view of the sleazy practice of firing Americans, and immediately replacing them with H-1Bs. However, this is not a consequence at all of the H-1B program. It is a consequence of the revival, since Reagan, of the old-fashioned doctrine "employment at will." Perhaps employment at will should be reexamined: perhaps because of the difference in power between the corporation and the employee, we should say that the employee is free to leave, but that the corporation, upon offering permanent employment, perhaps after a probationary period, must jump through hoops to let people go. The situation right now is absurd, for the free exercise of employment at will means that there is no difference between contract work and permanent employment, except for health insurance. But H-1B is the wrong fight for this, and it makes programmers to appear as mind-

lessly xenophobic.

Labor movements need to completely and explicitly disassociate themselves with racism and xenophobia, and programmers as any kind of labor movement need either to clam up about H-1B or call for the sort of economic borders that existed in the 19th century...probably the latter. In James Cameron's movie Titanic, the young fellow is able to board the doomed ship without any formalities after winning the price of a ticket in a card game. This was the reality in Europe and in America before the First World War and now that Communism is dead there really is no reason why we should not allow American programmers to work in India and vice-versa.

Another ugly feature about part of the anti-H-1B program is a sort of programming anti-intellectualism. This surfaced in my Internet flame war on this topic last May. I was told at one point by a poster that programming is not rocket science or mathematical, and for this reason, the story went, I was wrong in my claim that the high levels of mathematical skill found in Indian and Russian education made a difference. I was told that "it's just some reports, and some Access screens."

I was struck by the way that this sounded like management when management and sales have overcommitted to the customer. They've promised an innovative, ground-breaking system in no time flat and they really don't care about the "mere coders" who wind up burned-out, divorced, and suicidally depressed after months of 16 hour days. The "coders" protest that the real problem is complex and that their adventures in the forest of logic have validity: management says

Continued

## The C++ Class Implementation Checklist

By John Miano

Over the years I have had fix a lot of C++ code that was not done right the fix time around. Most of the time when I am assigned to look for ,problems I find signs of something wrong that leap off the screen at me. In order to try to get things right the first time, I have compiled various checklists over the years listing these signs of problem in order to help others identify problems at the initial stages of coding. Here I present one of these checklists, intended for C++ class implementation. The items covered in this checklist are those aspects of a class that are found within the header file.

### THE CHECKLIST

#### BEGINNINGS

1. Determine the type of class (normal, mixin, concrete, container)

#### REQUIRED FUNCTIONS

2. Has a default constructor
3. Has a virtual destructor unless a concrete class
4. Has a copy constructor (May be private)
5. Has an assignment operator (May be private)
6. Has a print to ostream function.

#### STRUCTURE

7. Consistent naming for classes, member functions, and member variables.
8. Mixin base classes declared virtual
9. No private virtual functions
10. Overloaded operators have conventional meanings
11. Declarations from other namespaces are qualified
12. Exception classes are derived from `std::exception`

#### CONVERSION

13. Constructors with one parameter are declared explicit unless intended for type conversion.
14. Cast operators are only to built-in types.

#### MEMBER FUNCTIONS

15. Member functions are declared const when they do not modify the state of the object.
16. No exception specifications unless required to derive from a class in the standard library.
17. Containers should contain both const and non-const functions for accessing elements.

#### MEMBER VARIABLES

18. Variables referenced by multiple threads are declared volatile.
19. No public member variables
20. STL classes used for array members
21. No bit fields

#### RETURN VALUES

22. Member functions do not return pointers or references to member variables unless the class is a container.
23. Do not use `char *` for string parameters and return values
24. If the return value is a condition code, possible return values are documented.
25. All functions return references rather than a pointers unless
  - o The function transfers ownership of the object referenced by the pointer.
  - o The function can return a `NULL` object.
26. If the function returns a status it should be through an integer function value.

#### C++ IDIOMS

27. String and member variables and parameters use string classes rather than `char *`.
28. `iostreams` are used in place of `FILE` pointers.
29. `register` modifier is not used.

#### PARAMETERS

30. Class parameters use references rather than pointers unless:
  - o The function transfers ownership of the object referenced by the pointer to the object.
  - o The parameter can be `NULL`.
31. Reference and pointer member function parameters are declared const if they are not modified.
32. Default parameter values supply the most common argument values and are not a substitute for overloading.

### AMPLIFIED CHECKLIST

#### BEGINNINGS

1. Determine the type of class (normal, mixin, concrete, container)

The first step in analyzing a class is to identify what the class is used for. No one set of rules can apply to all types of classes so for this checklist so I identify classes as concrete, container, mixin, and normal types. These are general groupings and are not mutually exclusive. For example a container frequently is a concrete type.

A concrete type is one not intended to be derived from

to create other classes. Examples of possible concrete types include string and complex number classes. The most notable attribute of a concrete class is that there are no virtual functions at all.

Container classes are ones intended to hold other objects. As such, containers may provide access to members not permitted for other types of objects. Other types of classes may exhibit container-like behavior.

Mixin classes are those intended only to add properties to other classes. Mixin classes are usually associated with multiple inheritance.

Normal classes are those classes that do not fall into the other special categories. Most classes fall into this category.

## **REQUIRED FUNCTIONS**

### 2. Has a default constructor

Without a default constructor, it is impossible to create arrays of a class. In some cases, a default constructor may not be appropriate in a class. Not having a default constructor should be the result of a design, not accident.

### 3. Has a virtual destructor unless a concrete class

If a class does not have a virtual constructor, using a pointer to a derived class to delete an object will call the wrong destructor.

```
class Base
{
public:
    ~Base ;
};
class Derived : public Base
{
};
Base *base = new Derived ;
delete base ; // BAD calls Base::~Base not Derived::~Derived.
```

### 4. Has a copy constructor (May be private)

### 5. Has an assignment operator (May be private)

The C++ compiler generates a default assignment operator and copy constructor for classes that do not have these functions defined. These defaults can produce unpredictable behavior if the class contains pointers, including member objects containing pointers. All classes should define the copy constructor and assignment operator, even if these are private declarations with no implementation.

```
class SomeClass
{
private:
    SomeClass (const SomeClass &) ;
    SomeClass &operator=(const SomeClass &) ;
};
```

### 6. Has a print to ostream function.

I find it useful to have a print function defined for classes for debugging.

```
class SomeClass
{
public:
    virtual void printOn (std::ostream &strm) const ;
};
```

By defining the << operator like this

```
inline operator std::ostream &operator<<(std::ostream &strm, const SomeClass
&object)
{
    object.printOn (strm) ;
    return strm ;
}
```

the class and all classes derived from it can be written to a stream in the usual manner.

## **STRUCTURE**

### 7. Consistent naming for classes, member functions, and member variables.

I use

- o Mixed case for class names (e.g. SomeClass)

- o Mixed case with initial lower case for member functions (e.g. someFunction)

- o All lower case with underscores for member variables (e.g. some\_variable)

- o All lower case for function variables (e.g. somevariable)

- o Mixed case for global functions (e.g. SomeFunction)

Use whatever scheme you like, but make it consistent.

I cannot be too strong in discouraging the practice of trying to encode a variable's type within its name.

### 8. Mixin base classes declared virtual

The use of mixin classes implies the use of multiple inheritance.

### 9. No private virtual functions

Private virtual functions make no sense.

### 10. Overloaded operators have conventional meanings

The ability to override operators makes it possible to define types with natural behaviors. For example, standard arithmetic operators are defined for matrices and complex numbers. When you overload operators keep in mind that + means addition or concatenation. - Means subtraction. Assigning other meanings to these, or any other operators, results in confusing code. Use named functions for all other operations.

Do not overload the && or || operators for any reason.

### 11. Declarations from other namespaces are qualified

Placing "using namespace" in header files defeats the purpose of namespaces. Use namespace qualified names instead within class definitions.

### 12. Exception classes are derived from std::exception

If all exceptions in an application have a common base class it is possible to catch all throw exceptions and identify them.

```
catch (std::exception &ee)
{
    cout << ee.what () ;
}
```

## **CONVERSION**

### 13. Constructors with one parameter are declared explicit

unless intended for type conversion.

With a constructor and function declared like these

```
class SomeClass
{
public:
    SomeClass (unsigned int size) ;
};
void SomeFunction (SomeClass object) ;
```

the constructor clearly is not intended to convert an unsigned int to a SomeClass like this:

```
SomeFunction (10) ; // Obviously an error but results in
// SomeFunction (SomeClass (10))
```

To guard against unintended conversion, declare constructors with one parameter as explicit.

```
explicit SomeClass (unsigned int size) ;
```

14. Cast operators are only to built-in types.

Implement type conversion only through constructors to avoid ambiguous conversions. For built-in types, the only way to implement conversion is through conversion operators and ambiguity is not an issue.

## **MEMBER FUNCTIONS**

15. Member functions are declared const when they do not modify the state of the object.

The const specification allows the compiler to determine which member functions modify the state of an object. This knowledge allows the compiler to make optimizations. The const declaration is easy to use, helps document the code and helps enforce the designed behavior of the code.

When in doubt, make it const.

16. No exception specifications unless required to derive from a class in the standard library.

In my humble opinion, exception specifications are the most useless feature added to the language in the C++ standard. They require much effort to implement but provide marginal benefit. In fact, if you ensure that all exception classes are derived from `std::exception`, exception specifications provide no value whatsoever.

Unfortunately, you will have to include an exception specification to create your own exception classes.

```
class MyException : public std::exception
{
public:
    virtual const char *what () const throw () { return "My exception" ; }
};
```

If you disagree with me and believe that exception specifications have value, you need to ensure that you use them consistently in all of your classes.

17. Containers should contain both const and non-const functions for accessing elements.

Without a const mechanism to access the contents of a container, it is impossible to access the contents of a const Container. To ensure that a function such as this is legal

```
void PrintContainer (const &SomeContainer &object)
{
    for (unsigned int ii = 0 ; ii < object.getSize () ; ++ ii)
```

```
        cout << object [ii] << endl ;
    }
```

you must have a const assessor function like this

```
class SomeContainer
{
    SomeObject &operator [] (unsigned int) ;
    Const SomeObject &operator [] (unsigned int) const ;
};
```

## **MEMBER VARIABLES**

18. Variables referenced by multiple threads are declared volatile.

Variables that can be modified by mechanisms not defined by the C++ standard should be declared as `VOLATILE`. The most likely reason for needing volatile variables is the use threads.

19. No public member variables

Allowing member variables to be accessed directly violates the principles of encapsulation. Once applications start referencing member variables it becomes impossible to change the behavior of the class, such as retrofitting it to incorporate thread synchronization, without changing all the code that uses it.

There is no advantage to allowing direct access to member variables. If efficiency is a concern, inline functions are just as fast.

20. STL classes used for array members

Pointers are evil. Instead of using pointers for dynamic arrays

```
int *int_array ;
int_array = new int [size] ;
memset (int_array, size * sizeof (int), 0) ;
```

use the standard template library.

```
std::vector<int> int_array ;
int_array.resize (size, 0) ;
```

and let the library manage memory for you.

21. No bit fields

The implementation of bit fields is system and compiler-specific. If you need a specific bit layout use an integer variable and bit operators to access the individual bits.

## **RETURN VALUES**

22. Member functions do not return pointers or references to member variables unless the class is a container.

Access member variables through member functions. Returning pointers or references to members make it impossible to enforce this restriction.

23. Do not use `char *` for string parameters and return values

Pointers are evil. Use `std::string` instead.

24. If the return value is a condition code, possible return values are documented.

25. All functions return references rather than a pointers unless

o The function transfers ownership of the object refer-

enced by the pointer.

- o The function can return a NULL object.

Pointers are evil.

26. If the function returns a status it should be through an integer function value.

Functions that can fail should either throw exceptions or return a status value. Since the return value should allow the caller to determine the cause of an error without having to call an additional function `bool` is not suitable as a status type.

## **C++ IDIOMS**

27. String and member variables and parameters use string classes rather than `char *`.

Pointers are evil.

28. `iostreams` are used in place of `FILE` pointers.

29. `register` modifier is not used.

Leave optimization to the compiler.

## **PARAMETERS**

30. Class parameters use references rather than pointers unless:

- o The function transfers ownership of the object referenced by the pointer to the object.

- o The parameter can be NULL.

31. Reference and pointer member function parameters are declared `const` if they are not modified.

Using `const` makes it clear which parameters are input only.

32. Default parameter values supply the most common argument values and are not a substitute for overloading.

When converting an integer to a string, the radix is usually 10. This is an acceptable use of a default parameter value.

```
std::string IntToString (int value, unsigned int radix = 10) ;
```

To implement searching by last name or by last name and first name use overloading

```
int GetCustomerId (std::string &lastname) ;  
int GetCustomerId (std::string &lastname, std::string &lastname) ;
```

not a default parameter value specifying a missing argument.

```
int GetCustomerId (std::string &lastname, char *firstname = 0) ;
```

---

Continued from Opinion

"it's just some reports and some screens" and completely dismisses adventures in logic as "not having the big picture." And I've noted that for the most part the Indian and Russian programmers with whom I have worked don't say "it's just some reports, and some Access screens." They are too busy solving the actual problem.

I taught some classes in the C programming language a

few years ago to assembler programmers in Chicago. I had students from Russia and native born Americans. I made an effort to communicate an essential fact about C, which is the concept of the Lvalue by means of which assembler language pointers enter this higher-level language. Some of the Americans complained that I used too much math, like Willy Loman in Arthur Miller's *Death of a Salesman*: when his son Biff Loman can't get into college because of failure at the math exam, Willy, finding that it's not all "a smile and a shoeshine", cries "math, math, math!" I have been told with a straight face by supervisors that programming has nothing to do with math, which is an interesting view. The Russians, on the other hand, liked the class and asked for an advanced C session.

In my view, math and logic is excellent preparation for programming because it allows you to predict sensibly what your code will do. To say "it's just some reports, it's just cut and paste, it's just some screens" is to willfully ignore what goes on under the covers and hope for the best. Sure, you can use Crystal Reports, but you still are responsible for setting up the codes, in VBA, for developing all calculated fields. If the Crystal Report report merely lists the data one for one, developing it is something the user can do on her lunch hour. And to say "it's just some Access screens" ignores the fact that most corporate development departments are abandoning Access in favor of SQL Server 7 at warp speed and moving programming logic into stored procedures...and that stored procedures are more than declarative SQL: they use yet another programming language, Transact-SQL.

In the past few years, I have done a lot of contracting and in almost every development environment I have seen the following occur. Some programmer is given the job of developing a routine with the usual silly deadlines, and in the usual absence of mentoring he develops Visual Basic that sends SQL strings directly to ADO. Some other programmer, usually with a name like Manu, Vladimir, or Devi, sets our boy straight and shows him how to use a stored procedure...or quietly reworks our boy's code.

I have also heard, over and over again, the following song and dance about objects: objects are a passing management fad, objects don't work, etc. In actual fact, once you've written a non-object function with twenty parameters, objects are the next logical step, because they replace impossible-to-maintain parameter lists and UDTs with simple assignments to properties. And it is true that objects, if designed without analysis, don't work, for it's untrue that any old object is as good as another. Objects, like

databases, have to be normalized: but in the real world many objects were apparently designed by Rube Goldberg, the original American programmer.

In my experience, many offshore programmers are more highly skilled at developing objects because their creativity has a bit of old fashioned discipline inculcated by their educational system. Traditional educational systems had the virtue and the vice of destroying false and valid self-esteem: but in the 1960s, in America, self-esteem became the most important goal of our educational system. Our own educational system encourages kids to think that if they created something it must be great. This is a half-assed preparation for life on the job: for in the real world, it is simply untrue that if you make something it is great: sometimes, it stinks. At the same time, if you don't innovate, there is simply no reason to keep you on the payroll. When I first developed a program which altered its own code, I was dazzled by my brilliance. Only later did I realize the downside...without losing the initial blast of self-esteem.

I have also heard and read the complaint that H-1Bs are used to discriminate against older programmers. I don't have any hard numbers on this issue, but my experience as a 50-year old seems to indicate very little age discrimination against males. The law of employment discrimination excepts "bona fide occupational qualifications": this is why guys with hairy tummies can't work at Hooters: since the Hooters corporation defines "the Hooters Experience" as having nice ladies serve beer in running shorts to a mostly male clientele, they can probably get away with gender discrimination. The only area in which bona-fide occupational qualifications do not apply is race discrimination, for a very good reason having to do with people like Dr. King.

Age is not a bona-fide occupational qualification, or disqualification, for a programmer position: but some of the resistance to change that is associate with age is. If a programmer insists on writing C++ code as if it were C (using malloc instead of new and doing everything in one main routine) he is not qualified. If a programmer is proud of bringing servers to their needs, by writing dynamic SQL in a three-tier environment, he is not qualified. If he does this because of his age, and his unwillingness to learn, too bad. Being able to write objects, being able to strongly type pointers, and knowledge of the three-tier way are all examples of what corporations in good faith require. I know that corporations do not always act in good faith, but this is an area where they do.

Cases of real discrimination exist, usually against older women who are indeed being excluded from the field because younger men can't manage women who look like their hippie Mom.

Offshore programmers and other guest workers have transformed and revitalized whole sections of our society in addition to the computing profession. Devon Avenue in Chicago used to be nearly abandoned and is now lively with restaurants, temples and mosques. Indian programmers in particular come from a proud tradition of respect for learning and they don't have the anti-intellectualism which in our profession generates crude, and impossible to maintain, code. Therefore I cannot get on board with the anti-H-1B movement. I don't pity the poor immigrant, I learn from him.