Do not change program semantics in the debug build

devblogs.microsoft.com/oldnewthing/20060815-00

August 15, 2006

Raymond Chen

What you don't want is a problem that goes away when you debug it.

It is expected that a program have additional debugging code inside `#ifdef DEBUG` blocks. After all, that's why it's a debug build. But what you definitely don't want to do is have that debugging to fundamentally change the program's behavior. You can perform additional validation. You can raise assertion failures. You can track resources. It can be slower and consume additional resources, but you had better not alter code flow.

```
// This is wrong
BOOL DoSomething(Thing *p)
{
#ifdef DEBUG
 // Do some extra parameter checking
 if (!p) {
  Log("Error! p parameter must not be NULL.");
  return FALSE; // WRONG!
 }
#endif
  ... remainder of function ...
}
```

This code is wrong: The debug version behaves fundamentally differently from the retail version. If somebody calls this function with `NULL` for the `p` parameter, the retail version of the program will crash but the debug build will trap the error and fail the call.

Do not change the function's semantics in the debug build. If the retail build crashes, then the debug build must also crash in the same way. Sure, you can log the failure before you crash, but you still need to crash.

An analogous mistake in the C# world might go like this:

```
// This is wrong
void DoSomething()
{
#if DEBUG
  try {
#endif
   ... guts of function ...
#if DEBUG
  } catch (Exception ex) {
     LogException(ex);
  }
#endif
}
```

In this C# example, the debug build logs and swallows exceptions, while the retail version allows them to escape.

If you mess up and write code like this, where the retail and debug versions behave in some fundamentally different way, you will eventually get yourself into this situation: The retail version has some problem, but the debug version works okay. Your customer can't figure out what the difference is, so they switched to the debug version on their production servers. It runs twice as slow and consumes three times as much memory, requiring significant capital expenses to scale up to their previous level of service. But it's the best they can do because the problem doesn't occur on the debug version (and therefore cannot be debugged there).

I have seen reports of software getting into this predicament, and it reflects very poorly on the developers.

Raymond Chen

Follow