Tony Hoare

From: Michael Hotchin
Sent: 25 March 2002 22:45
To: Daniel Doubrovkine; David Richter; Peter Shier; Michael Grier; Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

Importance: Low

At the other end of the 'utility' spectrum, when I worked on IceCAP, we used ASSERT's very liberally. In portions of the code ASSERT and other debug checking code approach 1/3 of the total code.

In almost *every* case, an ASSERT firing was a defect that had to be fixed. The testers thought they were great, our ASSERT macro would dump a stack trace to the clipboard, with that and the input data they had everything they needed to create really good bug reports.

We decided that having a framework for allowing a diversity of non-shipping actions was simply too useful to *not* have, so we came up with several mechanisms that would allow us to control the extra information and checking that the non-ship builds would process.

For example, I created a class that would dump info to the OutputDebugString port, but *only* if an ASSERT triggered during its lifetime. So, if there's something that's good to know if an ASSERT happens, but otherwise would just flood you with useless spammage, into an ASSERTINFO it would go, just as a stack variable.

It was particularly useful in our BVT lab - in case of an ASSERT failure, that lab software could collect both the stack trace *and* the output from DEBUGMON, and the tester / dev could sift though the debris after the fact. If I was still doing this today, I might have the ASSERT code create a memory dump as well, so things could be debugged after-the-fact.

I think it all comes down to the groups attitudes. In IceCAP a 'noisy' ASSERT simply would not be tolerated - it counted as a build break, and if you owned that ASSERT, you could expect several unhappy dev's and testers to visit you to persuade you either fix the condition or remove the ASSERT. For us, quality was driven into the product from day one - bugs were fixed as they were found, with none of this 'wait until code complete' nonsense that some groups adopt to give the illusion of forward progress.

Mike H.

-----Original Message-----
From: Daniel Doubrovkine
Sent: Monday, March 25, 2002 1:14 PM
To: David Richter; Peter Shier; Michael Grier; Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

There was this one product called Netdocs that used to assert in all directions during the development process. It was hell and developers hated it. People ended up not running debug builds that popped up windows all over the place with asserts that you could 'safely ignore'.
That was the worst example of how this great intention can make your life miserable.

> -----Original Message-----
> **From:** David Richter
> **Sent:** Monday, March 25, 2002 12:44 PM
> **To:** Peter Shier; Michael Grier; Tony Hoare
> **Cc:** Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
> **Subject:** RE: Why don't best practices get adopted?
>
> I disagree with this statement:
>
> > Either you care about the condition or you don't. If you do, then IMHO it should be a runtime check with appropriate error handling.

In order to expose errors during testing, particularly, to expose those errors closer in time to the root cause, test-only asserts may do checking that you do not want to normally run in retail code. For example, code that passes along a data structure might do minimal checking in retail code (e.g., just checking the header) and extensive checking in test mode (e.g., walking the entire data structure).

My understanding of best practice here is to leave such extensive checks in the retail build, but disabled in such a way that they can be enabled to help track down customer issues on the customer's box.

-----Original Message-----
From: Peter Shier
Sent: Monday, March 25, 2002 12:30 PM
To: Michael Grier; Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

I never use ASSERTs as I believe that any condition worth checking should also be checked in a retail build. These checks boil down to two types of conditions:

1) Internal logic errors
2) External conditions out of your control

I have seen lots of code with things like ASSERT(ptr != NULL). What is the point of that? Do you trust the pointer? If not, then check it in all builds.

In the XP Embedded infrastructure I used something very similar to Michael's INTERNAL_ERROR_CHECK all over the place and it paid off extremely well. If the specified condition exists, then there is a logic error in my code that should never happen so I have to bring down the app right now. For conditions that exist due to external errors I return an appropriate error code.

Some may argue that there are conditions that are not worth checking in retail builds because the code path will definitely be exercised in debug builds and the ASSERT will allow for early and clear error detection. I find that approach to be very over-confident. You would have to be 100% sure that all code and data paths through that ASSERT will be exercised in the debug builds and that their behavior will be exactly the same in retail builds. I would never say that about even the simplest line of code such as x = 5.

Either you care about the condition or you don't. If you do, then IMHO it should be a runtime check with appropriate error handling.

Peter

-----Original Message-----
From: Michael Grier
Sent: Monday, March 25, 2002 8:59 AM
To: Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

I agree with all your observations. At some level an assertion macro has common unquestionable behavior - evaluate an expression and if not true, take some action.

It's the "take some action" part that I think getting unified will be like corraling cats (to use someone else's clever expression).

For example, in my team I took a clear stance, which was operationally based. I leave it up to the programmer to determine which situation it is and then choose the appropriate condition checking. Our main assert macro could better be called "DETECT_FALSE_CONDITION_THAT_YOURE_WILLING_TO_BE_CALLED_IN_THE_MIDDLE_OF_THE_NIGHT_FOR_WHEN_THE_BUILD_BREAKS_AND_LEAVE_IGNORED_IN_RETAIL_BUILDS()".

Given how many bogus/"overactive" assertions there are in the overall product, I believe that the usual standard (no criticism implied) is more along the lines of "detect some unusual condition that I think shouldn't happen but which it's safe to ignore and make progress".

We ended up splitting assertions into two classes: hard and soft. Hard assertions are the

traditional "stop the production line for in place immediate debugging" type of thing. Soft assertions are just loud debug spew that includes text asking whoever happens to read it to log a bug on the issue, with a flag to turn them into hard assertion failures at runtime. Neither ship in retail, but we have a separate macro, "INTERNAL_ERROR_CHECK()" which is a lot like ASSERT in semantics, but in addition to having the hard assert break-in behavior in CHK builds, in retail builds, it causes propagation of ERROR_INTERNAL_ERROR in retail builds. Our QA team logs debug spew during our regression tests and logs bugs about any soft assertion failures reported and we fix them prior to checkin just like other assertion failures.

Maybe at some level this is the traditional abstraction naming problem - does the name describe what the function does or why you would call it? My conclusions is that base primitives ("non-interfaces" in my taxonomy) should be named based on what their implementation does while names in formal interfaces should be named based on intent; thus ASSERT == "if false, break the build" in my dev team. (This is legal in this realm since it's reasonable to have "well known" synonyms for the primitive behavior - notably use of the term "assert", "verify" and "smart" all have clear enough implementation meanings [hopefully] that it's reasonable to use them instead of writing "CDeleteWhenLeaveScope<Foo> pFoo;" we can write "CSmartPtr<Foo> spFoo;".

mjg

-----Original Message-----
From: Tony Hoare
Sent: Monday, March 25, 2002 1:48 AM
To: Michael Grier
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

That's the wrong hope! The real beauty of an assertion failure is that its meaning is adaptable to circumstances.
In a recent survey, I found about a dozen different meanings, and I am still looking for more. Can you help me?

In the simplest case, and assertion means a diagnostic dump in test; it is compiled out and ignored in ship code. Or it may send a stack dump back to Redmond. Other useful meanings are:

An assertion may be a guess about the behaviour of legacy code: violation is reported and ignored on test.

An assertion may be a simplifying assumption, to be removed before ship. On test, the test case is ignored. In ship code, it causes the compiler to object.

An assertion may be a precondition on a method call. Its failure is reported at the call site.

An assertion may be a postcondition. Its failure is reported at the method declaration.

An assertion may be an invariant, i.e., both a precondition and a postcondition.

An assertion may be a guide to a program analysis tool like PREfix, and help to reduce false positives.

In future, assertions may be used to help analysis to find true negatives; they may be used in diagnosis of dumps, they will be the basis for defect classification (as in Office Watson); and they may even be used by a compiler for code optimisation.

In summary, there may be about twenty useful meanings of assertion failure. I should be delighted to hear of more ideas. But not a thousand, I hope!

Yours, Tony.

-----Original Message-----
From: Michael Grier
Sent: 23 March 2002 03:46
To: Jay Krell; Tony Hoare

Cc: Steve Palmer; Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

Is there any actual hope that we can even agree on what an assertion failure means?

Heavyweight process is something we need to avoid at all costs unless we want to turn into yet another big inflexible software company (there's a great argument that we're already too big and inflexible in Windows and our ability to innovate has been massively restricted...)

On the other hand, I really think that there should be someone in charge of identifying some best practices and they should have the ability to more or less enforce the engineering teams to adopt these practices. Unfortunately this is like political office - the people who actually want this kind of position are the ones who by no means should ever be allowed to hold it. <half-grin> The mere existance of this organization/post will attract people who want to make arbitrary restrictions and build big process.

mjg

-----Original Message-----
From: Jay Krell
Sent: Friday, March 22, 2002 7:39 PM
To: Michael Grier
Subject: FW: Why don't best practices get adopted?

-----Original Message-----
From: Tony Hoare
Sent: Friday, March 22, 2002 6:47 AM
To: Steve Palmer
Cc: Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

You are right (I've got it tight this time, haven't I?). Anyway, your comment shows a very sober judgment.

I can corroborate your experience in the case of assertion macro's. Every development guide that I have seen recommends a set that is private to one team. In Windows, a recent count revealed over a thousand different assert macro **declarations**. The only large team with a standardized assert macro is Office, and the Office Watson tool relies on this to classify defects in RAID.

The proliferation of build processes among development teams has absorbed an enormous effort from the PREfast developers. Is there any hope of unifying the notations for small range of assertion types, so that the tool developers can begin to exploit them?

Thanks for the correction and the contribution.

Yours, Tony.

-----Original Message-----
From: Steve Palmer
Sent: 15 March 2002 22:52
To: Grant George; Tony Hoare; Marco Dorantes Martinez; Rick Andrews; Productivity Discussions-CodeDevelopment
Cc: Six Sigma; Six Sigma Code Development Managers; Edgar Alberto Herrador Nieto
Subject: RE: Why don't best practices get adopted?

A best practice that comes in the form of a tool developed by another group may not be sufficiently supported in terms of troubleshooting, upgrades or documentation beyond that group to warrant any other group

adopting it. The exceptions, of course, are tools such as PREfast and PREfix that have a support structure within MSR. The same applies to tools developed by the Productivity Tools Team.

-----Original Message-----
From: Grant George
Sent: Friday, March 15, 2002 8:58 AM
To: Tony Hoare; Marco Dorantes Martinez; Rick Andrews; Productivity Discussions-CodeDevelopment
Cc: Six Sigma; Six Sigma Code Development Managers; Edgar Alberto Herrador Nieto
Subject: RE: Why don't best practices get adopted?

I have seen a number of obstacles in my experience (pre and post coming to Microsoft) to adopting best practices:
1) The not-invented-here problem. Personal pride and glory can sometimes get in the way of adopting a smarter solution that comes from the outside versus one you might be crafting locally in your team.
2) Culture and process co-dependency. Many of the best practices that work for a particular team don't translate well to a team with a different culture. The process in Team A might be well-honed and refined over time and fit that team's approach to testing or development or program management, but doesn't work as well in a different group.
3) Timing. Sometimes best practices are bound up with a specific tools and the ability to move from a legacy environment to a new tool environment comes only at certain windows of the product development schedule and if you don't make that shift at the right time, you lose the opportunity.
4) Age. The longer a product group using a particular methodology with some dependent tools has been around and doing things the same way, the harder it is to unseat the old practice (and related tools) when you want to make a shift.
5) Basic championship. Often a superior practice to one currently being used does not get adopted without support from the top leadership of the group and a torch-bearer to champion it. That torch bearer must also be well respected within the group culture and seen as one who can affect change without alienating folks, and has a personal (read: review goals) stake in the shift.
6) Trade-off. Too often process and tool shifts are approached as an all-or-nothing proposition and that's both myopic and insensitive to each team's unique characteristics. A number of initiatives over the years we have pursued in Office (spec inspections, common ui test automation tools, test library management solutions, etc) each have slight twists (call it a local accent to the mother tongue) in each team that allows for some shades-of-gray implementation locally that still fits into the overall common best practice or tool set. You have to allow for and plan for this from the outset.

-----Original Message-----

From:	Tony Hoare
Sent:	Friday, March 15, 2002 8:34 AM
To:	Marco Dorantes Martinez; Rick Andrews; Productivity Discussions-CodeDevelopment
Cc:	Six Sigma; Six Sigma Code Development Managers; Edgar Alberto Herrador Nieto
Subject:	RE: Why don't best practices get adopted?

You are tight. If you want some really significant case studies, you should look at projects in which first use of a new technology has been successful, but for which the technology was abandoned in all successor projects, even by the original team members. The reason for this cannot just be ignorance or fear of the new.

I have been involved in two such projects, when as an academic I was collaborating with Industry. In both of them, the initial trial

project was well chosen as suitable for the new technology, and the participants were self-selected enthusiasts, who understood the technology well enough to adapt it further to the needs of that project. On successful completion of the initial project, it did not seem that the next project was sufficiently similar to the successful one that there could be a guarantee that the same techniques will be successful again.

A second factor is that the original team is now dispersed through a somewhat larger project. The other members of the new teams are no longer so keen to undertake something new (which anyway is not so original any more), and they feel it might be beyond their powers. They would need a couple of weeks' training to give them the necessary skills and confidence. But there is no time. The project is already behind schedule, and it would be managerially unacceptable at this stage to accept a two-week delivery delay. (Actually, in one case at least, failure to use the new technology led to much more serious delays at a later stage)

The lesson I have drawn from these two sad stories is that our only hope lies in the development of automatic programming and testing tools. As they evolve in the light of user requests and suggestions, they act as an accumulation of the experience of all who have used them in the past, and they immediately propagate the benefits to all who use them now. My stories date from the eighties, when good tools were conspicuously absent, and we had to rely on exhortation, education, and good will. These are good enough to generate a small band of early enthusiasts, but they cannot adequately influence the majority. And managerial edict is known to be dangerous at worst and ineffective at best.

Have you given any thought of how your best practices could be supported by tools? An obvious target would be an extension of PREfast, but perhaps we should also think of tools that assist in the planning of test harnesses and the generation of test cases. They could certainly contribute to your goal of getting the test strategy planned early.

Tony Hoare.

-----Original Message-----
From: Marco Dorantes Martinez
Sent: 14 March 2002 21:09
To: Rick Andrews; Productivity Discussions-CodeDevelopment
Cc: Six Sigma; Six Sigma Code Development Managers; Edgar Alberto Herrador Nieto
Subject: RE: Why don't best practices get adopted?

Hi,

This topic is of paramount importance, thanks for doing this kind of job and for conducting this discussion thread.

Also, the topic is a very complex one, because it is about how creative and ingenious people (programmers) work, and some times these guys are not very well in articulating their thoughts in clear and unambiguous statements (well, that is human behavior any way).

In my experience, I see some factors that prevent/permit quick adoption of best practices,
Here I am going to start with one of them, productivity/time, that is, the amount of work done by unit of time, something like velocity of development.

The practices that get my attention very quickly are those that help

me get things done faster and with high-quality.
I have found that those practices enclose sound principles, methods and rules about software design and programming, and also integrate well with related practices across the entire software development process.

For a general study on related subjects see the following classics:

"The Psychology of Computer Programming: Silver Anniversary Edition"
by Gerald M. Weinberg
ISBN: 0-932-63342-0
Dorset House

"Peopleware : Productive Projects and Teams, 2nd Ed."
by Tom Demarco, Timothy Lister
ISBN: 0-932-63343-9
Dorset House

"The Mythical Man-Month, Anniversary Edition : Essays on Software Engineering"
by Frederick P. Brooks
ISBN: 0201835959
Addison-Wesley

Best regards,
Marco

> -----Original Message-----
> **From:** Rick Andrews
> **Sent:** Thursday, March 14, 2002 2:17 PM
> **To:** Productivity Discussions-CodeDevelopment
> **Cc:** Six Sigma; Six Sigma Code Development Managers
> **Subject:** Why don't best practices get adopted?
>
> Hi,
>
> I'm doing a study to try to determine why Best Practices don't get adopted; and would love your input.
>
> In my work, I've seen several best practices being followed by teams here at Microsoft, but rarely do I see other teams adopt them – even though some have been measured with proven results. When I discuss this with the teams that have the best practices, they tell me they've shared their practices with others but still they don't get adopted. Sure, some get adopted (e.g., PREFast); I'm trying to understand the issues that drive adoption and learn why some get adopted and others don't.
>
> What's the secret to get a team to change their existing processes and adopt a proven best practice? What prevents teams from doing so? What's the key to getting individuals to adopt best practices?
>
> If you have any thoughts, ideas, suggestions, etc., I'd really appreciate your feedback. Feel free to reply privately, or to the alias if you think it's a suitable discussion topic. If you want your response to be confidential, please be sure to clearly tell me that. Finally, I'd be happy to meet, 1-1 or in a group, with anyone interested in discussing this further face to face. If you prefer to meet and can't find time available, feel free to schedule me for any morning from 9-10am where my schedule shows me as tentative.
>
> I'm also looking for volunteers to help with this study (e.g., collect data, analyze data, brainstorm, etc.). If you'd like to

participate, please contact me.

I'll publish my results on http://dev and http://SixSigma/CodeDevelopment when the study/analysis is complete. I'd like to get everyone's input by 3/22 (sooner the better) and have the results ready by 3/29.

Thanks for your help!

Rick Andrews

****

Rick Andrews
Six Sigma Blackbelt/SDE

SixSigma Team, http://SixSigma/CodeDevelopment
Concerned about Coding Quality? If so, join the Productivity Discussions - Coding alias (ProdDisC) to learn more and participate in helping ship higher quality software to our customers. Click here to join ProdDisC.

A bad day working is better than a good day reworking!

<< OLE Object: Picture (Metafile) >>

Tony Hoare

From: Joe Porkka
Sent: 25 March 2002 21:13
To: Peter Shier
Cc: Productivity Discussions-CodeDevelopment
Subject: RE: Why don't best practices get adopted?

I've got to disagree here - including all assertions in retail build for internal logic errors is unreasonable.
Certainly, asserts get abused (for handling your case 2: external conditions for example) - but including all assertions in retail builds will just discourage people from using them. If I know that assertions won't affect my retail size/performance then I don't hesitate to use them. If I had to worry that they may impact performance then I'm going to be a lot more conservative about using them. For example:

```
void MySortFunction(List *pList)
{
   // Some nifty fast sorting code goes here...
   ...
   ...
   // end sorting code.
   Assert( IsSorted(pList)); // EXPENSIVE test to ensure the list is really sorted.
}
```

In this example, if the assertion failed it may not be possible to recover - there is a serious flaw in the code and potentially corrupted data already.
If I lived in a world where Assert() didn't go away in retail builds, then I would delete that line of code as soon as I had verified MySortFunction() works correctly.

> I have seen lots of code with things like ASSERT(ptr != NULL). What is the point of that? Do you trust the pointer? If not, then check it in all builds.

It's not about trust - it's about validating and documenting your assumptions and requirements (among other reasons). Preconditions and Postconditions are excellent uses of assertions. Preconditions let you be very explicit and clear what conditions must be met for a piece of code to execute correctly. Postconditions let you very explicitly say what the possible range of results are. This is invaluable information for users, maintainers and debuggers of a piece of code.

> -----Original Message-----
> **From:** Peter Shier
> **Sent:** Monday, March 25, 2002 12:30 PM
> **To:** Michael Grier; Tony Hoare
> **Cc:** Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
> **Subject:** RE: Why don't best practices get adopted?
>
> I never use ASSERTs as I believe that any condition worth checking should also be checked in a retail build. These checks boil down to two types of conditions:
>
> 1) Internal logic errors
> 2) External conditions out of your control
>
> I have seen lots of code with things like ASSERT(ptr != NULL). What is the point of that? Do you trust the pointer? If not, then check it in all builds.
>
> In the XP Embedded infrastructure I used something very similar to Michael's INTERNAL_ERROR_CHECK all over the place and it paid off extremely well. If the specified condition exists, then there is a logic error in my code that should never happen so I have to bring down the app right now. For conditions that exist due to external errors I return an appropriate error code.
>
> Some may argue that there are conditions that are not worth checking in retail builds because the code path will definitely be exercised in debug builds and the ASSERT will allow for early and clear error detection. I find that approach to be very over-confident. You would have to be 100% sure that all code and data paths through that ASSERT will be exercised in the debug builds and that their behavior will be exactly the same in retail builds. I would never say that about even the simplest line of code such as x = 5.
>
> Either you care about the condition or you don't. If you do, then IMHO it should be a runtime check with appropriate error handling.
>
> Peter

-----Original Message-----
From: Michael Grier
Sent: Monday, March 25, 2002 8:59 AM
To: Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

I agree with all your observations. At some level an assertion macro has common unquestionable behavior - evaluate an expression and if not true, take some action.

It's the "take some action" part that I think getting unified will be like corraling cats (to use someone else's clever expression).

For example, in my team I took a clear stance, which was operationally based. I leave it up to the programmer to determine which situation it is and then choose the appropriate condition checking. Our main assert macro could better be called "DETECT_FALSE_CONDITION_THAT_YOURE_WILLING_TO_BE_CALLED_IN_THE_MIDDLE_OF_THE_NIGHT_FOR_WHEN_THE_BUILD_BREAKS_AND_LEAVE_IGNORED_IN_RETAIL_BUILDS()".

Given how many bogus/"overactive" assertions there are in the overall product, I believe that the usual standard (no criticism implied) is more along the lines of "detect some unusual condition that I think shouldn't happen but which it's safe to ignore and make progress".

We ended up splitting assertions into two classes: hard and soft. Hard assertions are the traditional "stop the production line for in place immediate debugging" type of thing. Soft assertions are just loud debug spew that includes text asking whoever happens to read it to log a bug on the issue, with a flag to turn them into hard assertion failures at runtime. Neither ship in retail, but we have a separate macro, "INTERNAL_ERROR_CHECK()" which is a lot like ASSERT in semantics, but in addition to having the hard assert break-in behavior in CHK builds, in retail builds, it causes propagation of ERROR_INTERNAL_ERROR in retail builds. Our QA team logs debug spew during our regression tests and logs bugs about any soft assertion failures reported and we fix them prior to checkin just like other assertion failures.

Maybe at some level this is the traditional abstraction naming problem - does the name describe what the function does or why you would call it? My conclusions is that base primitives ("non-interfaces" in my taxonomy) should be named based on what their implementation does while names in formal interfaces should be named based on intent; thus ASSERT == "if false, break the build" in my dev team. (This is legal in this realm since it's reasonable to have "well known" synonyms for the primitive behavior - notably use of the term "assert", "verify" and "smart" all have clear enough implementation meanings [hopefully] that it's reasonable to use them instead of writing "CDeleteWhenLeaveScope<Foo> pFoo;" we can write "CSmartPtr <Foo> spFoo;".

mjg

-----Original Message-----
From: Tony Hoare
Sent: Monday, March 25, 2002 1:48 AM
To: Michael Grier
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

That's the wrong hope! The real beauty of an assertion failure is that its meaning is adaptable to circumstances.
In a recent survey, I found about a dozen different meanings, and I am still looking for more. Can you help me?

In the simplest case, and assertion means a diagnostic dump in test; it is compiled out and ignored in ship code. Or it may send a stack dump back to Redmond. Other useful meanings are:

An assertion may be a guess about the behaviour of legacy code: violation is reported and ignored on test.

An assertion may be a simplifying assumption, to be removed before ship. On test, the test case is ignored. In ship code, it causes the compiler to object.

An assertion may be a precondition on a method call. Its failure is reported at the call site.

An assertion may be a postcondition. Its failure is reported at the method declaration.

An assertion may be an invariant, i.e., both a precondition and a postcondition.

An assertion may be a guide to a program analysis tool like PREfix, and help to reduce false positives.

In future, assertions may be used to help analysis to find true negatives; they may be used in diagnosis of dumps, they will be the basis for defect classification (as in Office Watson); and they may even be used by a compiler for code optimisation.

In summary, there may be about twenty useful meanings of assertion failure. I should be delighted to hear of more ideas. But not a thousand, I hope!

Yours, Tony.

-----Original Message-----
From: Michael Grier
Sent: 23 March 2002 03:46
To: Jay Krell; Tony Hoare
Cc: Steve Palmer; Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

Is there any actual hope that we can even agree on what an assertion failure means?

Heavyweight process is something we need to avoid at all costs unless we want to turn into yet another big inflexible software company (there's a great argument that we're already too big and inflexible in Windows and our ability to innovate has been massively restricted...)

On the other hand, I really think that there should be someone in charge of identifying some best practices and they should have the ability to more or less enforce the engineering teams to adopt these practices. Unfortunately this is like political office - the people who actually want this kind of position are the ones who by no means should ever be allowed to hold it. <half-grin> The mere existance of this organization/post will attract people who want to make arbitrary restrictions and build big process.

mjg

-----Original Message-----
From: Jay Krell
Sent: Friday, March 22, 2002 7:39 PM
To: Michael Grier
Subject: FW: Why don't best practices get adopted?

-----Original Message-----
From: Tony Hoare
Sent: Friday, March 22, 2002 6:47 AM
To: Steve Palmer
Cc: Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

You are right (I've got it tight this time, haven't I?). Anyway, your comment shows a very sober judgment.

I can corroborate your experience in the case of assertion macro's. Every development guide that I have seen recommends a set that is private to one team. In Windows, a recent count revealed over a thousand different assert macro **declarations**. The only large team with a standardized assert macro is Office, and the Office Watson tool relies on this to classify defects in RAID.

The proliferation of build processes among development teams has absorbed an enormous effort from the PREfast developers. Is there any hope of unifying the notations for small range of assertion types, so that the tool developers can begin to exploit them?

Thanks for the correction and the contribution.

Yours, Tony.

Tony Hoare

From: Michael Grier
Sent: 25 March 2002 20:38
To: Brian Manthos; Peter Shier; Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

This is what our PARAMETER_CHECK macro does. There's a global setting for whether parameter validation failures break in or not. What we did wrong is that there should be PARAMETER_CHECK and PARAMETER_CHECK_INTERNAL, and PARAMETER_CHECK_INTERNAL always breaks in on CHK builds since it reflects a bad internal caller. The PARAMETER_CHECK breaking in behavior should also be governed by the use of the app verifier. Maybe. Subject to the other endless debates about how good we have to be to bad callers.

mjg

-----Original Message-----
From: Brian Manthos
Sent: Monday, March 25, 2002 12:38 PM
To: Peter Shier; Michael Grier; Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?
Importance: Low

I take it you've never used this construct?

```
HRESULT SomeFunc( ... )
{
    if ( ... )
    {
        // We shouldn't be seeing this with our own sample
        // client code.  If so, we're setting a bad example.
        ASSERT(false);
        return E_INVALIDARG;
    }

    ...
}
```

✉ Brian

-----Original Message-----
From: Peter Shier
Sent: Monday, March 25, 2002 12:30 PM
To: Michael Grier; Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

I never use ASSERTs as I believe that any condition worth checking should also be checked in a retail build. These checks boil down to two types of conditions:

1) Internal logic errors
2) External conditions out of your control

I have seen lots of code with things like ASSERT(ptr != NULL). What is the point of that? Do you trust the pointer? If not, then check it in all builds.

In the XP Embedded infrastructure I used something very similar to Michael's INTERNAL_ERROR_CHECK all over the place and it paid off extremely well. If the specified condition exists, then there is a logic error in my code that should never happen so I have to bring down the app right now. For conditions that exist due to external errors I return an appropriate error code.

Some may argue that there are conditions that are not worth checking in retail builds because the code path will definitely be exercised in debug builds and the ASSERT will allow for early and clear error detection. I find that approach to be very over-confident. You would have to be 100% sure that all code and data paths through that ASSERT will be exercised in the debug builds and that their behavior will be exactly the same in retail builds. I would never say that about even the simplest line of code such as x = 5.

Either you care about the condition or you don't. If you do, then IMHO it should be a runtime check with appropriate error handling.

Peter

-----Original Message-----
From: Michael Grier
Sent: Monday, March 25, 2002 8:59 AM
To: Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

I agree with all your observations. At some level an assertion macro has common unquestionable behavior - evaluate an expression and if not true, take some action.

It's the "take some action" part that I think getting unified will be like corraling cats (to use someone else's clever expression).

For example, in my team I took a clear stance, which was operationally based. I leave it up to the programmer to determine which situation it is and then choose the appropriate condition checking. Our main assert macro could better be called "DETECT_FALSE_CONDITION_THAT_YOURE_WILLING_TO_BE_CALLED_IN_THE_MIDDLE_OF_THE_NIGHT_FOR_WHEN_THE_BUILD_BREAKS_AND_LEAVE_IGNORED_IN_RETAIL_BUILDS()".

Given how many bogus/"overactive" assertions there are in the overall product, I believe that the usual standard (no criticism implied) is more along the lines of "detect some unusual condition that I think shouldn't happen but which it's safe to ignore and make progress".

We ended up splitting assertions into two classes: hard and soft. Hard assertions are the traditional "stop the production line for in place immediate debugging" type of thing. Soft assertions are just loud debug spew that includes text asking whoever happens to read it to log a bug on the issue, with a flag to turn them into hard assertion failures at runtime. Neither ship in retail, but we have a separate macro, "INTERNAL_ERROR_CHECK()" which is a lot like ASSERT in semantics, but in addition to having the hard assert break-in behavior in CHK builds, in retail builds, it causes propagation of ERROR_INTERNAL_ERROR in retail builds. Our QA team logs debug spew during our regression tests and logs bugs about any soft assertion failures reported and we fix them prior to checkin just like other assertion failures.

Maybe at some level this is the traditional abstraction naming problem - does the name describe what the function does or why you would call it? My conclusions is that base primitives ("non-interfaces" in my taxonomy) should be named based on what their implementation does while names in formal interfaces should be named based on intent; thus ASSERT == "if false, break the build" in my dev team. (This is legal in this realm since it's reasonable to have "well known" synonyms for the primitive behavior - notably use of the term "assert", "verify" and "smart" all have clear enough implementation meanings [hopefully] that it's reasonable to use them instead of writing "CDeleteWhenLeaveScope<Foo> pFoo;" we can write "CSmartPtr<Foo> spFoo;".

mjg

-----Original Message-----
From: Tony Hoare
Sent: Monday, March 25, 2002 1:48 AM
To: Michael Grier
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

That's the wrong hope! The real beauty of an assertion failure is that its meaning is adaptable to circumstances.
In a recent survey, I found about a dozen different meanings, and I am still looking for more. Can you help me?

In the simplest case, and assertion means a diagnostic dump in test; it is compiled out and ignored in ship code. Or it may send a stack dump back to Redmond. Other useful meanings are:

An assertion may be a guess about the behaviour of legacy code: violation is reported and ignored on test.

An assertion may be a simplifying assumption, to be removed before ship. On test, the test case is ignored. In ship code, it causes the compiler to object.

An assertion may be a precondition on a method call. Its failure is reported at the call site.

An assertion may be a postcondition. Its failure is reported at the method declaration.

An assertion may be an invariant, i.e., both a precondition and a postcondition.

An assertion may be a guide to a program analysis tool like PREfix, and help to reduce false positives.

In future, assertions may be used to help analysis to find true negatives; they may be used in diagnosis of dumps, they will be the basis for defect classification (as in Office Watson); and they may even be used by a compiler for code optimisation.

In summary, there may be about twenty useful meanings of assertion failure. I should be delighted to hear of more ideas. But not a thousand, I hope!

Yours, Tony.

-----Original Message-----
From: Michael Grier
Sent: 23 March 2002 03:46
To: Jay Krell; Tony Hoare
Cc: Steve Palmer; Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

Is there any actual hope that we can even agree on what an assertion failure means?

Heavyweight process is something we need to avoid at all costs unless we want to turn into yet another big inflexible software company (there's a great argument that we're already too big and inflexible in Windows and our ability to innovate has been massively restricted...)

On the other hand, I really think that there should be someone in charge of identifying some best practices and they should have the ability to more or less enforce the engineering teams to adopt these practices. Unfortunately this is like political office - the people who actually want this kind of position are the ones who by no means should ever be allowed to hold it. <half-grin> The mere existance of this organization/post will attract people who want to make arbitrary restrictions and build big process.

mjg

-----Original Message-----
From: Jay Krell
Sent: Friday, March 22, 2002 7:39 PM
To: Michael Grier
Subject: FW: Why don't best practices get adopted?

-----Original Message-----
From: Tony Hoare
Sent: Friday, March 22, 2002 6:47 AM
To: Steve Palmer
Cc: Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

You are right (I've got it tight this time, haven't I?). Anyway, your comment shows a very sober judgment.

I can corroborate your experience in the case of assertion macro's. Every development guide that I have seen recommends a set that is private to one team. In Windows, a recent count revealed over a thousand different assert macro **declarations**. The only

large team with a standardized assert macro is Office, and the Office Watson tool relies on this to classify defects in RAID.

The proliferation of build processes among development teams has absorbed an enormous effort from the PREfast developers. Is there any hope of unifying the notations for small range of assertion types, so that the tool developers can begin to exploit them?

Thanks for the correction and the contribution.

Yours, Tony.

Tony Hoare

From: Michael Grier
Sent: 25 March 2002 16:59
To: Tony Hoare
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

I agree with all your observations. At some level an assertion macro has common unquestionable behavior - evaluate an expression and if not true, take some action.

It's the "take some action" part that I think getting unified will be like corraling cats (to use someone else's clever expression).

For example, in my team I took a clear stance, which was operationally based. I leave it up to the programmer to determine which situation it is and then choose the appropriate condition checking. Our main assert macro could better be called "DETECT_FALSE_CONDITION_THAT_YOURE_WILLING_TO_BE_CALLED_IN_THE_MIDDLE_OF_THE_NIGHT_FOR_WHEN_THE_BUILD_BREAKS_AND_LEAVE_IGNORED_IN_RETAIL_BUILDS()".

Given how many bogus/"overactive" assertions there are in the overall product, I believe that the usual standard (no criticism implied) is more along the lines of "detect some unusual condition that I think shouldn't happen but which it's safe to ignore and make progress".

We ended up splitting assertions into two classes: hard and soft. Hard assertions are the traditional "stop the production line for in place immediate debugging" type of thing. Soft assertions are just loud debug spew that includes text asking whoever happens to read it to log a bug on the issue, with a flag to turn them into hard assertion failures at runtime. Neither ship in retail, but we have a separate macro, "INTERNAL_ERROR_CHECK()" which is a lot like ASSERT in semantics, but in addition to having the hard assert break-in behavior in CHK builds, in retail builds, it causes propagation of ERROR_INTERNAL_ERROR in retail builds. Our QA team logs debug spew during our regression tests and logs bugs about any soft assertion failures reported and we fix them prior to checkin just like other assertion failures.

Maybe at some level this is the traditional abstraction naming problem - does the name describe what the function does or why you would call it? My conclusions is that base primitives ("non-interfaces" in my taxonomy) should be named based on what their implementation does while names in formal interfaces should be named based on intent; thus ASSERT == "if false, break the build" in my dev team. (This is legal in this realm since it's reasonable to have "well known" synonyms for the primitive behavior - notably use of the term "assert", "verify" and "smart" all have clear enough implementation meanings [hopefully] that it's reasonable to use them instead of writing "CDeleteWhenLeaveScope<Foo> pFoo;" we can write "CSmartPtr<Foo> spFoo;".

mjg

-----Original Message-----
From: Tony Hoare
Sent: Monday, March 25, 2002 1:48 AM
To: Michael Grier
Cc: Jay Krell; Productivity Discussions-CodeDevelopment; Rick Andrews; Steve Palmer; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

That's the wrong hope! The real beauty of an assertion failure is that its meaning is adaptable to circumstances. In a recent survey, I found about a dozen different meanings, and I am still looking for more. Can you help me?

In the simplest case, and assertion means a diagnostic dump in test; it is compiled out and ignored in ship code. Or it may send a stack dump back to Redmond. Other useful meanings are:

An assertion may be a guess about the behaviour of legacy code: violation is reported and ignored on test.

An assertion may be a simplifying assumption, to be removed before ship. On test, the test case is ignored. In ship code, it causes the compiler to object.

An assertion may be a precondition on a method call. Its failure is reported at the call site.

An assertion may be a postcondition. Its failure is reported at the method declaration.

An assertion may be an invariant, i.e., both a precondition and a postcondition.

An assertion may be a guide to a program analysis tool like PREfix, and help to reduce false positives.

In future, assertions may be used to help analysis to find true negatives; they may be used in diagnosis of dumps, they will be the basis for defect classification (as in Office Watson); and they may even be used by a compiler for code optimisation.

In summary, there may be about twenty useful meanings of assertion failure. I should be delighted to hear of more ideas. But not a thousand, I hope!

Yours, Tony.

-----Original Message-----
From: Michael Grier
Sent: 23 March 2002 03:46
To: Jay Krell; Tony Hoare
Cc: Steve Palmer; Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

Is there any actual hope that we can even agree on what an assertion failure means?

Heavyweight process is something we need to avoid at all costs unless we want to turn into yet another big inflexible software company (there's a great argument that we're already too big and inflexible in Windows and our ability to innovate has been massively restricted...)

On the other hand, I really think that there should be someone in charge of identifying some best practices and they should have the ability to more or less enforce the engineering teams to adopt these practices. Unfortunately this is like political office - the people who actually want this kind of position are the ones who by no means should ever be allowed to hold it. <half-grin> The mere existance of this organization/post will attract people who want to make arbitrary restrictions and build big process.

mjg

-----Original Message-----
From: Jay Krell
Sent: Friday, March 22, 2002 7:39 PM
To: Michael Grier
Subject: FW: Why don't best practices get adopted?

-----Original Message-----
From: Tony Hoare
Sent: Friday, March 22, 2002 6:47 AM
To: Steve Palmer
Cc: Productivity Discussions-CodeDevelopment; Rick Andrews; Marco Dorantes Martinez; Grant George
Subject: RE: Why don't best practices get adopted?

You are right (I've got it tight this time, haven't I?). Anyway, your comment shows a very sober judgment.

I can corroborate your experience in the case of assertion macro's. Every development guide that I have seen recommends a set that is private to one team. In Windows, a recent count revealed over a thousand different assert macro **declarations**. The only large team with a standardized assert macro is Office, and the Office Watson tool relies on this to classify defects in RAID.

The proliferation of build processes among development teams has absorbed an enormous effort from the PREfast developers. Is there any hope of unifying the notations for small range of assertion types, so that the tool developers can begin to exploit them?

Thanks for the correction and the contribution.