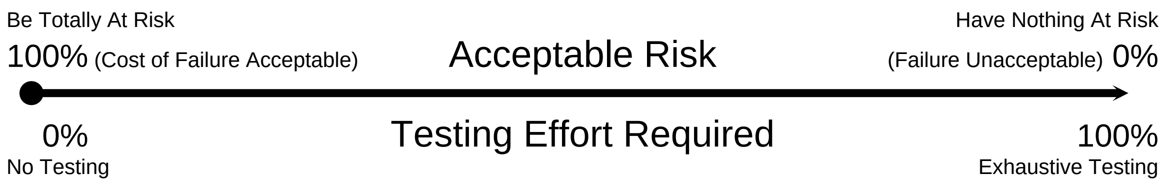
# When to Regression Test

Software testing is about *risk*.  The goal of testing is to make an accurate and fair assessment of the product based upon reproducible observations which allow management to make an *informed* business decision to deploy, or not, when weighed against the *acceptable level of risk* the company is willing to burden.



Achieving a 100% tested product is realistically impossible.  Testing is expensive.  However, defects discovered after deployment are far more costly than those found prior to deployment. The objective is to do *enough* testing that the *risk* of finding a post-deployment defect becomes acceptable.  Put another way, test until the cost of repairing the defect is *less* than finding it via testing.

The art of testing is to find that optimum point, based on the number of suspected defects remaining.  It is always better to err on the side of more testing when that point appears as a grey-area.

Testing organizations have two ways to move that point further to the right without incurring hits against cost or resources: look for ways to test faster and look for ways to avoid extensive testing where defects are not suspected (allowing the remaining time for other important tests).

*Automated regression testing* does both.

New code[1] often reveals more defects than previously tested old code.  Testing organizations, not just developers, leverage code subroutines, modules, and interfaces.  Once trust has been established for a piece of code, it is possible to move forward with other testing endeavors, such as examining dependent code.

It is important to note that just because code has been tested and passed doesn't *certify* it as error-free forever.  A new test might be developed, the environment might change, a new set of inputs may be used, or even the code base, resources, or services that *it* depends on may change.  A simple one-line code change, patch, or software tweak can have cascading results, sometimes not even noticed by the module where the change was made.

Consequently whenever *anything* changes, it is appropriate to do a regression test to revalidate the trust in the software.

When a full-scale regression test is too costly, *partial regression testing* may be employed as long as the *associated risks are well understood*.  This paper explores when it is appropriate to do a *partial* regression test versus a *full* regression test.

[1] Surprisingly, studies show that errors are 20% more likely to appear in code that has been just touched to repair a defect. [McConnell]  In other words, one-fifth of code repairs introduce *new* errors.  This is an important statistic when one realizes that 50% or more of the organization's time is frequently devoted to error detection and removal.  [Kit, p. 26]

First let's cover some basic definitions.

**Incremental Testing.**

1. To test *only* the code that was added or modified against *anticipated* results.
2. [Temporal.] To repeatedly test in a scheduled set of increments.[2] For example, after the nightly build.

**Incremental Regression Testing.**

Selective retesting of portions of the software that has changed, or is impacted by a change, against a known baseline of results to verify that there are no unintended side effects.

**Full Regression Testing.**

Complete retesting of the entire software application against a known baseline of results to verify that there are no unintended side effects anywhere in the application.

Regression testing can be conducted at the unit, integration, and system levels and may apply to both functional and non-functional testing.

Good testers are capable of implementing and designing more tests than would ever be practical to implement or execute. Exhaustive testing means products would never ship or be very expensive. Therefore, testing should focus on the right things, not wasting limited resources on the lesser important things. [Kit]

While we know we can't test everything, and compromises must be made, effort expended on the tests that were created should be leveraged. Because of inter-code dependency, defect detection today leads to defect prevention tomorrow. Testing early and resolving issues prevent defects from migrating downstream where they become more costly. Therefore regression testing should be integrated *into* the development cycle, and not treated as a phase after completion.

In an *ideal world* you should not have to regression test the *whole system* just because of minor maintenance. [Weide] And, it is possible to perform successful incremental regression testing, but this *requires* extensive examination of the code to understand which modules/blocks/etc. are affected by code modifications and which test cases address them. [Agrawal, *paraphrased*.]

Most people bypass regression testing because they think their change is innocent. [McConnell]

Full testing is mandatory for *critical* and *risk-aversive* applications. Partial regression testing is tolerable for small, non-critical software that has a captive user population.

To engage in incremental regression testing, trace matrices *must* be used for test coverage. [Pierce] And, it is important to note that the degree of regression testing required for a system is not proportional to the failure corrected, but to the extent of the change applied to correct the failure. [Hoover] Test case selection *may* work if the function call graph is known, or the software design is has *full* traceability – often this requires complex analysis tools and can be more costly than just running a full regression test. Arbitrarily omitting test cases used in regression testing is risky. [Gadgil] It is exceedingly difficult to perform analysis on the ripple effect of a change. This is why it is common in Extreme Programming practice to rerun *all* tests several times a day. [Binder]

That said, let's examine what it takes to do incremental regression testing, sometimes called modular or partial regression testing.

[2] While this practice is strongly endorsed, it is not the focus of this whitepaper; therefore we will be using the other definition for our purposes of discussion.

Assuming that there is full traceability and the software's design is well understood, regression tests should be triggered whenever:

- a new subclass has been developed
- a superclass has changed
- a server class has changed
- a defect has been resolved
- a new system build has been generated
- a new increment is generated for system scope integration testing or system testing
- a system has stabilized and the final release build has been generated

The regression tests that should be selected are those that [Binder / Tsai]:

- exercise critical code to the system
- address code that is costly to fix or deploy[3] later
- feature code which implements frequently used features
- exercise the new code or modified code
- address fixes of defects
- address performance increases
- exercise components in which testing found defects the *last* regression test
- exercise portions of the system in which defects were found *after* testing (by the customer– meaning testing wasn't adequate in the first place)
- has functional dependency shared with another piece of code
- has an input and/or output dependency that is shared with another piece of code
- has persistent data
- has execution dependence on another component, system, module, or middleware
- has conditional dependence on another component, system, or module
- would be impacted by middleware changes

Code can be exempt from regression testing,[4] if its observable behavior does not depend on *any* static data values kept privately within the component and meets none of the above conditions.

Defining the modular boundary impacted by any given change is laborious, and once identified *all* components within the boundary *must* be regression tested together. Realistically, you *need* to look at potentially *the entire system* to check for weird interactions. If modular regression testing actually were sound for "real" software, there would only be unit testing. [Weide]

It is nearly impossible to produce a high-quality software product unless you can systematically retest it after changes have been made. The only practical way to manage regression testing is to automate it. [McConnell] If you don't run regression tests regularly, you may discover that the application broke due to a code change made three months ago. [Hunt]

While *theoretically* possible to perform incremental regression testing, to do it properly *requires extensive analysis, bookkeeping, and additional process discipline*. In the real world, it is simpler and always more effective to perform an automated full regression test in order to obtain a trustable assessment of the product.

At an absolute minimum, *always* run a full suite of regression tests against any full system build that you intend to deploy, dazzle at tradeshows, show the press, or wish to flaunt in front of sales staff and their customers.

[3] Such as an EPROM in sealed MRIs spread across hospitals world wide.
[4] Not testing as a whole!

## Resources

Amland, Ståle. 1999. *Risk Based Testing and Metrics.* EuroSTAR.

Agrawal, Hiralal. *Incremental Regression Testing.*

Beck, Kent. 1999. *Extreme Programming Explained.* Addison-Wesley.

Binder, Robert. 1999. *Testing Object-Oriented Systems.* Addison-Wesley.

Collofello, James. 1996. *Modeling Software Testing Processes.* IEEE.

Gadgil, Harshawardhan. 2002. *A paper on Regression Testing Techniques.* Indiana Univ.

Hoover, Alex. 2003. *Software Regression Testing.*

Hunt, Andrew; et. al. 1999. *The Pragmatic Programmer.* Addison-Wesley.

Information Processing Ltd. 1996. *Why Bother to Unit Test?*

Jacobson, Ivar; et. al. 1998. *The Unified Software Development Process.* Addison-Wesley.

Kaner, Cem; et. al. 1999. *Testing Computer Software*. Wiley.

Kim, Jung-Min; et. al. *An Empirical Study of Regression Test Application Frequency.*

Kit, Edward. 1995. *Software Testing in the Real World*. Addison-Wesley.

McConnel, Scott. 1993. *Code Complete*. Microsoft Press.

Pierce, Preston. *Software Verification & Validation*. ISBN 0-7803-3277-6.

Rothman, Johanna. *What Does It Cost to Fix a Defect? — www.stickyminds.com*

Rothermel, Gregg; et. al. *A Safe, Efficient Algorithm for Regression Test Selection*. Clemson Univ.

Software QA / Test Resource Center — www.softwareqatest.com

Tsai, Wei-Tek; et. al. *Scenario-Based Functional Regression Testing.* Arizona State Univ.

Weide, Bruce. *“Modular Regression Testing”: Connections to Component-Based Software.* Ohio State Univ.

Wilson, Rodney. 1997. *Software Rx: Secrets of Engineering Quality Software.* Prentice Hall.

Wu, Ye; et. al. *Regression Testing on Object-Oriented Programs.*