OpenInheritance

Martin Fowler

tags: *encapsulation · API design*

This is the opposite attitude to DesignedInheritance. Advocates of open inheritance do not look to disallow inheritance by Sealing their classes or doing anything else to stop people inheriting classes.

Fans of open inheritance don't disagree with designed inheritance fans about the dangers of inheritance. Inhertiance is an initmate relationship with plenty of dangers for the unwary. You can easily break superclass behavior and you have to be expecially wary of upgrades since you can be relying on hidden implementation details. But open inheritors belive that the choice of whether to take on those risks should be the decision of the users of the library. If they choose to inherit, then they accept the risks and the consequences.

A designed approach is in theory better, but the problem is that it's very hard to figure out all the ways in which someone can usefully vary an existing class. This leads to two problems: library users can't use a library in a way that the designer didn't anticipate and library designers have to take on the responsibility of trying to anticipate extensions.

Designed inhertance assumes that library writers are wiser than their users. Often this isn't the case. Library writers make mistakes too, if inheritance is open then the user of the class has the opportuntity to fix them and still take advantage of the library writers hard work.

Designing for inheritance is a very tricky task, particularly as you usually don't know what the use cases are. Asking people to think through all inheritance cases places a big burden on library writers. Open inheritance means they don't have to deal with it unless they specfically want to, with the proviso that those that do inherit have to be careful about what they are doing.

One area that particularly causes trouble for designed inheritance is testing. Often with testing you need to use a TestDouble, but are prevented by a sealed class with no interface that you can use for substitution. These kinds of APIs can be Detestable

Open inheritors often do design classes with intended extension points that are safer to use than others. However they usually mark these through naming or documentation rather sealing them. That way people are pointed to the safe areas but not prevented from overriding elsewhere if necessary.

Often design inheritance comes with a DirectingAttitude, which usually forgets that fools are ingenious when it comes to messing things up.

tags

API design · academia · agile · agile adoption · agile history · analysis patterns · application architecture · application integration · bad things · big data · build scripting · certification · clean code · collaboration · conferences · continuous integration · data analytics · database · delivery · dictionary · diversions · diversity · documentation · domain driven design · domain specific language · domestic · encapsulation · evolutionary design · extreme programming · gadgets · internet culture · language feature · lean · legacy rehab · metrics · microsoft · mobile · noSQL · object collaboration design · parser generators · photography · presentations · process theory · productivity · programming platforms · project planning · projects · recruiting · refactoring · refactoring boundary · requirements analysis · retrospective · ruby · scrum · software craftsmanship · team environment · team organization · technical debt · technical leadership · testing · thoughtworks · tools · travel · uml · version control · web · website · writing
All Content

translations

Japanese · Spanish · Korean · Chinese · Thai

blogroll

ThoughtBlogs
TW Alumni · Tim Anderson · Kent Beck · Tim Bray · Paul Duvall · Elizabeth Hendrickson · Jeremy Miller · Ralph Johnson · Dan Pritchett · xkcd

Guides
Intro
Design
Agile
NoSQL
DSL
Delivery
About Me

Popular Articles
New Methodology
Dependency Injection
Mocks aren't Stubs
Is Design Dead?
Continuous Integration

Books
NoSQL Distilled
Domain-Specific Languages
Refactoring
Patterns of Enterprise Application Architecture
UML Distilled
Analysis Patterns
Planning Extreme Programming
Signature Series

Site Sections
FAQ
Content Index
Bliki
Books
DSL Catalog
EAA Catalog
EAA Dev
Photos

ThoughtWorks
Blogs
Careers
Mingle
Twist
Go