Issue 049: Object-Oriented Programming

Adrian Kosmaczewski

October 3rd, 2022

Welcome to the forty-ninth issue of *De Programmatica Ipsum*, dedicated to *Object-Oriented Programming*, featuring a brand new logo and visual style, and starting the 5th year (say what?) of publication of this magazine.

In this edition:

- We apply Gartner's Hype Cycle to OOP[1] and watch its evolution in the past half-century.
- We re-read "Design Patterns" by the Gang of Four[2] in the Library section[3].
- In the Vidéothèque section[4], we review a GOTO Berlin 2017 video by James Coplien[5].

We would also like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Christopher Nascone, Jean-Paul de Vooght, Patryk Matuszewski, Paul Hudson, and Roger Turner.

Enjoy this issue! Please subscribe to our free newsletter[6] to stay updated about new releases,

[1] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[2] https://deprogrammaticaipsum.com/the-gang-of-four/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/category/videotheque/
[5] https://deprogrammaticaipsum.com/james-coplien/
[6] https://deprogrammaticaipsum.com/newsletter/

share the articles on social media, or contribute[7] if you would like to support our work.

Cover photo by Mike Meyers[8] on Unsplash[9].

[7] https://deprogrammaticaipsum.com/contribute/

[8] https://unsplash.com/@mike_meyers?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[9] https://unsplash.com/s/photos/object?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

The Hype Cycle Of OOP

Adrian Kosmaczewski

October 3rd, 2022

Even though Marketing buzzwords might have an effect akin to Kryptonite against our readers, we are going to use the famous Gartner's Five-Step Hype Cycle[1] to take a closer look at the practice of Object-Oriented Programming (OOP) and its various ups and downs in the past 50 years. Remembering that one of the core tenets of this magazine is to make the impossible dialogue[2] possible, the framework provided by Gartner fits this task perfectly well.

Innovation Trigger

With a silent and barely noticed birth at the end of the 1960s, Simula 67[3] introduced OOP concepts to the world: classes, inheritance, polymorphism, garbage collection, and the whole package (no pun intended.) We say "barely" because Jean Sammet noticed and reported Simula in her masterpiece[4].

The following programming language to apply concepts of OOP (that we know of) was, of course, Smalltalk, which we covered extensively in a previous edition of this magazine[5]. In 2009 James Iry wrote about the birth of Smalltalk and OOP in one of the most popular posts ever published on Blogspot: "A Brief, Incomplete, and Mostly Wrong History of Programming Languages"[6]:

[1] https://www.gartner.com/en/research/methodologies/gartner-hype-cycle
[2] https://deprogrammaticaipsum.com/the-impossible-dialogue/
[3] https://en.wikipedia.org/wiki/Simula
[4] https://deprogrammaticaipsum.com/jean-sammet/
[5] https://deprogrammaticaipsum.com/the-absolute-no-frills-quite-ignorant-very-incomplete-and-certainly-flawed-beginners-guide-to-smalltalk/
[6] https://james-iry.blogspot.com/2009/05/brief-incomplete-and-mostly-wrong.html

> 1980 – Alan Kay creates Smalltalk and invents the term "object oriented." When asked what that means he replies, "Smalltalk programs are just objects." When asked what objects are made of he replies, "objects." When asked again he says "look, it's all objects all the way down. Until you reach turtles."

Simula 67 (a superset of ALGOL 60[7]) and Smalltalk famously appear in the "inspiration list" of almost all OOP programming languages created during the following 50 years, including C++, Java, C#, and many others.

OOP concepts took quite some time to take hold; at least 20 years of patient evangelization[8] and, to a certain extent, indoctrination. The famous August 1981 edition of Byte Magazine[9] featuring a colorful balloon (a symbol that would eventually become the official language logo) was quite an isolated hallmark. The market for hobbyist (and, to a large degree, enterprise) computing was not ready for OOP yet. We had to wait precisely five years, until the August 1986 issue[10], to find a significant reference to the world of OOP in the pages of Byte.

In the case of Dr. Dobb's Journal, there was the first mention of OOP in a review of NEON (a cross between Smalltalk and FORTH[11], the latter a language dear to Dr. Dobb's Journal in those days). It was in an article on page 96 of the October 1985 issue (volume 10, page 785[12]) explaining how to send messages to an object "Bic" of type "Pen" to move around the screen. There was an article dedicated to OOP in March 1987 (volume 12, page 241[13]) and, interestingly enough, featured in the column "Artificial Intelligence." The subject of OOP became much more popular afterward; suffice to mention March 1988's article "Object-Oriented Programming and Databases" (volume 13, page 119[14]) or November 1990's "Roll your own Object-Oriented language" (about Object Prolog, in volume 15, page 991[15])

Finally, the introduction to the collection of articles published by Dr. Dobb's Journal in 1989 (volume 14, page 13[16]) begins with a phrase that says it all:

> If there's any lingering impression of the past year — and that is reflected in this collection of DDJ articles — it is that in 1989 object-oriented programming stormed to the forefront in the world of programming.

Peak Of Inflated Expectations

OOP grew dramatically during the decade between 1985 and 1995. The first OOPSLA Conference[17] was held in 1986, chaired by none other than Daniel Ingalls of Smalltalk fame. Bjarne Stroustrup released the first versions of C++. Brad Cox[18] melted Smalltalk on top of C and created Objective-C. Bertrand Meyer released both Eiffel and its associated major bestseller book[19]. Borland added OOP support to Turbo Pascal[20] 5.5 (released May 1989).

[7] https://en.wikipedia.org/wiki/ALGOL_60
[8] https://deprogrammaticaipsum.com/less-evangelization-more-honestization/
[9] https://archive.org/details/byte-magazine-1981-08/
[10] https://archive.org/details/byte-magazine-1986-08
[11] https://en.wikipedia.org/wiki/Forth_(programming_language)
[12] https://archive.org/details/dr_dobbs_journal_vol_10/page/784/mode/2up
[13] https://archive.org/details/dr_dobbs_journal_vol_12/page/240/mode/2up
[14] https://archive.org/details/dr_dobbs_journal_vol_13/page/118/mode/2up
[15] https://archive.org/details/dr_dobbs_journal_vol_15/page/991/mode/2up
[16] https://archive.org/details/dr_dobbs_journal_vol_14/page/n13/mode/2up
[17] http://www.oopsla.org/oopsla-history/
[18] https://deprogrammaticaipsum.com/brad-cox/
[19] https://deprogrammaticaipsum.com/bertrand-meyer/
[20] https://en.wikipedia.org/wiki/Turbo_Pascal

Famous groups of experts like "The Three Amigos,"[21] "The Gang of Four,"[22] and the burgeoning Agile movement started coalescing around OOP, yielding concepts and acronyms that would change the shape of our work forever, such as Design Patterns, Test-Driven Development, Scrum, Extreme Programming, standup meetings, and finally the Agile Manifesto.

Agile and OOP are closely related to one another in terms of community and timing, as explained by James Coplien[23] (another prominent figure of both movements) in the Vidéothèque article this month[24].

By 1992 OOP was the next big thing™®©, as NeXT's marketing material conveniently reminded us[25], with a Nike-level slogan stating that *"The object is the advantage."*

OOP reached the pinnacle of its fame during the decade starting in 1995. The significant earthquake moment of that era was the release of Java[26], still one of the world's most popular programming languages, 26 years after its release. Those were the times of the standardization of C++ as ISO/IEC 14882:1998; the rise of the Design Patterns movement; the explosion of TDD[27] and xUnit frameworks; the introduction of .NET[28] by Bill Gates; the release of the "Head First"[29] series of books at O'Reilly; the spread of the idea of inversion of control and dependency injection[30].

And to top it off, all of these things happened at the same time as the rise of the World Wide Web, propelling OOP technologies to the spotlight as suitable ways to build and maintain web applications, with names such as WebObjects, Django, .NET, Ruby on Rails, and JavaBeans, making the headlines of blogs and magazines.

This author argues that the OOP frenzy lasted until January 2007.

Through of Disillusionment

During the late 2000s and early 2010s, OOP suffered various targeted attacks, lost credibility and popularity, and was almost unanimously regarded as an antipattern. Why did this happen?

Simultaneously to Steve Jobs's introduction of the iPhone in January 2007, Google started publishing a series of blog posts called "Testing on the Toilet,"[31] or "TotT" for short. On May 15th, 2008, this series featured a famous issue: "TotT: Using Dependancy Injection to Avoid Singletons."[32] The writing was literally on the wall of toilets worldwide: Singletons are bad™®©. Sadly, the much more exciting idea of dependency injection contained in the article got lost in the minds of most readers.

The reference to the iPhone in the previous paragraph is not anodyne: Cocoa Touch, the

[21] https://deprogrammaticaipsum.com/the-three-amigos-among-others/
[22] https://deprogrammaticaipsum.com/the-gang-of-four/
[23] https://en.wikipedia.org/wiki/Jim_Coplien
[24] https://deprogrammaticaipsum.com/james-coplien/
[25] https://www.youtube.com/watch?v=rf5o5liZxnA&t=2096s
[26] https://deprogrammaticaipsum.com/write-anywhere-run-once/
[27] https://deprogrammaticaipsum.com/kent-beck/
[28] https://www.zulenet.com/see/BillGatesNET.html
[29] https://deprogrammaticaipsum.com/kathy-sierra/
[30] https://www.martinfowler.com/articles/injection.html
[31] https://testing.googleblog.com/2007/01/introducing-testing-on-toilet.html
[32] https://testing.googleblog.com/2008/05/tott-using-dependancy-injection-to.html

main application framework used to create iPhone apps, featured various singletons[33] all over the place: `NSNotificationCenter`, `NSFileManager`, `NSWorkspace`, `UIApplication`, `UIAccelerometer` and the list continues... testability be damned. But, but, but Objective-C was dynamic, fans argued, so a savvy programmer could easily OCMock[34] their way into testable code. Yeah, maybe not. The truth is that Google's stance prevailed, fueled by an engineering culture many developers dreamt of joining.

And if Singletons were terrible, who knows what evils the other Design Patterns could be hiding?

Other authors blogged their concerns around OOP and Java (sometimes mostly, sadly, about Java): Paul Graham[35], Peter Norvig[36], Steve Jegge[37], and Joel Spolsky[38]. I wrote about those bad times in a previous article[39]:

> Fifteen years ago, Bruce Tate published "Beyond Java[40]." This title was the reaction against what seemed as a chokehold of Java and the JVM on the server side of the software development industry. Proposing alternatives such as Python, Smalltalk or Ruby, it remains as the book that spoke the zeitgeist of its era: the mid-2000's were not a good time for Java.

Conflating Java with OOP made things worse. Java was (and to a large degree, still is) the punching bag, the programming language we like to hate, no matter which other language we use every day. Moving away from Java became moving away from OOP.

During those days, we saw a revival of functional programming concepts, fueled by the prowesses and promises of "Big Data" and a newly found interest in machine learning. Many (if not all) popular OOP languages adopted `filter()`, `map()`, and `collect()` functions, conveniently encapsulating cumbersome `for` and `while` loops. C# went to the extreme of offering an extension called LINQ[41] that encapsulated those functional principles in a SQL-like syntax. Similarly, Java got sidelined by Scala on the server and later by Kotlin on the mobile client.

Programmers saw a new type of object emerge: closures. A callable entity to be passed around functions, closing around the values defined in the stack frame where it appeared. At least for Objective-C, a literal object and the only one allocated on the stack.

On the other side of the fence, and more or less at the same time, functional languages like F# mixed OOP concepts and naturally interacted with a sizeable pre-populated collection of classes initially intended to be consumed with C#.

The Model-View-Controller architecture, whose acronym became associated with Massive View Controller[42], gave way to similarly-named alternatives: Django's MVT[43], Swing's

[33] https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html
[34] https://ocmock.org/
[35] http://www.paulgraham.com/icad.html
[36] https://www.norvig.com/design-patterns/
[37] https://steve-yegge.blogspot.com/2007/12/codes-worst-enemy.html
[38] https://www.joelonsoftware.com/2005/12/29/the-perils-of-javaschools-2/
[39] https://deprogrammaticaipsum.com/write-anywhere-run-once/
[40] https://www.oreilly.com/library/view/beyond-java/0596100949/
[41] https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/
[42] https://akos.ma/blog/refactoring-ios-projects/#2-class-design-smells
[43] https://www.javatpoint.com/django-mvt

MVP[44], Microsoft's MVVM[45], Swift's VIPER[46], and more.

Design patterns also suffered during those days. Singleton was the first victim; then came Observer. It is sad because it was a fine idea, a helpful mechanism to decouple the Model from the rest of the MVC concept. In Cocoa Touch, the Observer pattern became (in)famous through `NSNotification` and Key-Value-Observing.

But since the release of Swift in 2014, the Observer pattern died and was replaced by Reactive programming, a more generic idea built around data streams, which found fertile ground in web programming. In particular, since Facebook and Google respectively released the first versions of React[47] (2013) and Angular[48] (2016). Professor Salvaneschi from the University of Saint Gallen (Switzerland) explained the road from the Observer design pattern to Reactive programming in a paper published in[49]2014[50].

Slope Of Enlightenment And Plateau Of Productivity?

We have reached the point of productivity, which is why OOP is no longer hype: it is boring, tested, and works. But to reach this stage, OOP has had to morph, losing some characteristics that were inextricably associated with it.

In particular, OOP lost inheritance along the way; in many cases, it lost classes altogether.

Look at JavaScript and its prototype-based inheritance; the .NET DLR[51] (Dynamic Language Runtime) and its DynamicObject[52] class. In a move that would make Coplien happy, OOP is once again about objects and no longer about classes.

On the other hand, we lost class inheritance altogether: multiple implementation inheritance (à la C++ and Eiffel) had already died when Java (and its multiple inheritances of interfaces, borrowed from Objective-C's protocols) became popular. The thing is, in the last decade, we also lost single inheritance: neither Go nor Rust[53] have it, and we are talking here of two of the most popular programming languages of the 2010s.

To be honest, multiple inheritance came back in the form of traits, allowing developers to compose objects with various personalities. And single inheritance, well, it can now be found in the most unlikely of places: Docker container images can happily "inherit" `FROM` one another, overriding parameters and configuration settings layer by layer. Good luck finding out which base image overrode which setting.

TDD pundits can always inject any dependencies they need; there is a staggering number of dependency injection frameworks, making everyone happy. Joel Spolsky[54] loves duct tape programmers, and he is right.

The market cherry-picked the best parts of OOP: encapsulation and messaging. The Cloud

[44] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter
[45] https://learn.microsoft.com/en-us/windows/communitytoolkit/mvvm/introduction
[46] https://www.raywenderlich.com/8440907-getting-started-with-the-viper-architecture-pattern
[47] https://en.wikipedia.org/wiki/React_(JavaScript_library)
[48] https://en.wikipedia.org/wiki/Angular_(web_framework)
[49] https://link.springer.com/chapter/10.1007/978-3-642-55099-7_7
[50] https://link.springer.com/chapter/10.1007/978-3-642-55099-7_7
[51] https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/dynamic-language-runtime-overview
[52] https://learn.microsoft.com/en-us/dotnet/api/system.dynamic.dynamicobject?view=net-6.0
[53] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/
[54] https://www.joelonsoftware.com/2009/09/23/the-duct-tape-programmer/

Native world coopted these two and remarketed them as microservices[55], exchanging little JSON messages sent over HTTP or through a message broker like Kafka or RabbitMQ. Kubernetes is the new object-oriented application runtime, and you can even have Aspect Oriented Programming[56] thanks to eBPF[57].

The tradeoff? In these new OOP systems, the latency is worse than using Objective-C's `objc_msgSend()` or COM+ `IDispatch` and even worse than using C++ vtables, but now we can safely apply Conway's law[58] and divide our teams conveniently: so that they only communicate using JSON messages on the cluster and with nerf guns on the open office floor plan.

Cover photo by Steve Johnson[59] on Unsplash[60].

[55] https://akos.ma/blog/microservices-or-not-your-team-has-already-decided/

[56] https://en.wikipedia.org/wiki/Aspect-oriented_programming

[57] https://ebpf.io/

[58] https://en.wikipedia.org/wiki/Conway's_law

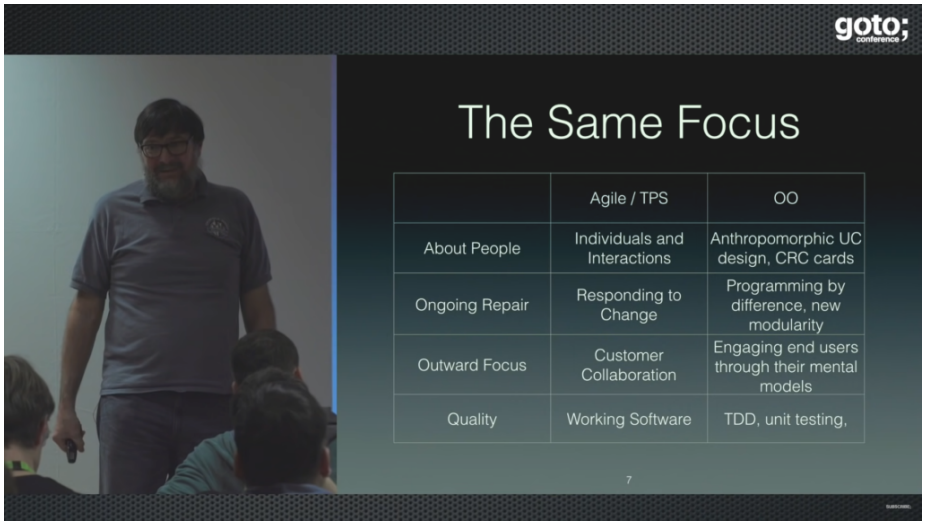
[59] https://unsplash.com/@steve_j?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[60] https://unsplash.com/s/photos/rejected?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

James Coplien

Adrian Kosmaczewski

October 3rd, 2022



For some shameful reason, most probably because of some profound ignorance on our side, we managed to publish four years of this magazine without mentioning James "Jim" ("Cope") Coplien not even once. It is high time to correct this, and there is no better moment to do that than in this issue dedicated to Object-Oriented Programming.

James Coplien has had a long relationship with Agile and OOP; he was Program Chair at OOPSLA 1996[1]; he gave a name to the "Curiously Recurring Template Pattern"[2] in C++; he studied Borland's software engineering[3] craftsmanship and quality; dared to mock[4] the Gang of Four[5] patterns; and he also wrote in 1991 what many consider to be one of the best books about C++ ever published: "Advanced C++: Programming Styles and Idioms."[6] His influence is such that Kent Beck[7], who defines himself in terms of relationships, claims[8] to have invented Extreme Programming based on Jim's ideas. The 1990s was an exciting time for object-oriented discussions; we were still far away from the times of the backlash[9].

A prolific author[10], we could easily have talked about James in the Library[11] section of this magazine; instead, we will focus our attention on a particular video. In GOTO Berlin 2017[12],

[1] http://www.oopsla.org/oopsla-history/
[2] https://en.wikipedia.org/wiki/Curiously_recurring_template_pattern
[3] https://www.researchgate.net/publication/2816856_Borland_Software_Craftsmanship_A_New_Look_at_Process_Quality_and_Productivity
[4] https://wiki.c2.com/?KansasCityAirConditioner
[5] https://deprogrammaticaipsum.com/the-gang-of-four/
[6] https://archive.org/details/advancedcbsprogr00copl/
[7] https://deprogrammaticaipsum.com/kent-beck/
[8] http://www.oopsla.org/2002/fp/files/spe-metahpor.html
[9] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[10] https://sites.google.com/a/gertrudandcope.com/www/books
[11] https://deprogrammaticaipsum.com/category/library/
[12] https://gotober.com/2017/

he gave a substantial overview of his thinking around Agile, OOP, testing, and programming in a now-classic video: "The Dehumanisation of Agile and Objects."[13]

The gist of the video stands in a single concept, whereby the Agile and Object Orientation movements were essentially born with the same goal: to channel human thinking into software artifacts as faithfully as possible. In essence, objects were a mechanism to model concepts, which is why the Agile Manifesto[14] stresses so much "Individuals and interactions over processes and tools."

It was all about humans. Yet... we got it all wrong; maybe, ironically enough, because we are humans. Instead of objects, we got classes; instead of Agile, we got SAFe; instead of building quality into our software, we got TDD.

Unbeknownst to many software developers, the work of MIT's Seymour Papert[15] is the origin of OOP, who based his research on the findings of Swiss child psychologist Jean Piaget[16]. A child psychologist. Now it all makes sense: the reason why Adele Goldberg[17] and Alan Kay built a tool named "Smalltalk"[18] is to be able to teach programming to kids[19]. Thus James Coplien asserts, and rightfully so, that "every product owner should be a UX person" (minute 44:05[20].) We wanted to allow kids to discover the world by themselves, using a Dynabook (not an iPad.) Instead, we have social networks "gamifying" and "monetizing" their interactions, feeding their brains with the image of a divided, unbearable[21], immutable world.

Having almost literally thrown the water with the babies, our industry insists on quarreling about programming language A versus B, multiple versus single inheritance, strong versus weak typing, tabs versus spaces, JSON versus YAML, or whether to curly brackets or not.

This video lets some desperation through; you have been warned. It has a long run of epic quotes; here are just a few:

> "If you are working on one class at a time, you don't even know where the link to the next object goes. By design you are not allowed to know where it goes because of this wonderful thing called *polymorphism*. It's GOTOs on steroids!"

(minute 15:39[22])

> "TDD is not a testing technique, it's a design technique."

(minute 24:07[23])

> "Our industry cannot afford to suck a little less; we need paradigm shifts."

(minute 29:49[24])

> "There's no DevOps teams in Scrum."

[13] https://www.youtube.com/watch?v=ZrBQmIDdls4
[14] https://agilemanifesto.org/
[15] https://en.wikipedia.org/wiki/Seymour_Papert
[16] https://en.wikipedia.org/wiki/Jean_Piaget
[17] https://deprogrammaticaipsum.com/adele-goldberg/
[18] https://deprogrammaticaipsum.com/the-absolute-no-frills-quite-ignorant-very-incomplete-and-certainly-flawed-beginners-guide-to-smalltalk/
[19] https://deprogrammaticaipsum.com/michael-hiltzik/
[20] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=2645s
[21] https://deprogrammaticaipsum.com/business-as-unusual/
[22] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=939s
[23] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=1447s
[24] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=1789s

(minute 34:40[25])

> (About Daily Scrums) "Now here you are, you're probably all in a Scrum team and you've been doing this a million times and you don't know what you're doing!"

(minute 35:34[26])

> "These conferences are doing unbelievable damage (...) and have driven both Agile and Object-Orientation into the ground."

(minute 35:46[27])

Finally, to the question, "I'm trapped working in a Java[28] shop, what should I do?" James replied with an Obama-style mic drop: "You should refactor... your CV." (minute 50:16[29]).

Besides the fun, Jim makes an interesting final observation about the things that are actually working well in the industry nowadays. These are regression and integration test automation, mob programming, refactoring organizational structures, and UX knowledge in general. Interestingly, he also mentions microservices (individual entities, developed and deployed separately, and most importantly, interacting via messaging) as a proper OOP paradigm, a point also raised by Graham in his book "OOP The Easy Way,"[30] published in 2018.

Jim Coplien ends his talk with a simple call to action: at every conference where new technology is introduced, software engineers should strive to be skeptical; to ask whether it improves the quality of life of society as a whole, and to take it home only to increase the human value of our products and services.

Focus on the people. Again and again. Because that is what Agile and OOP were all about, to begin with.

Graham explained in his AppBuilders 2018 talk[31], one that fits this edition of De Programmatica Ipsum perfectly well, that if monads are burritos[32] (are they?[33]), then objects are jam donuts[34]. Maybe we got Agile and OOP completely wrong.

Complement James Coplien's GOTO Berlin Conference video with a discussion[35] between him and Robert "Uncle Bob" Martin at the 2007 JAOO Conference[^1] about TDD. In it, Jim explained his disagreement with Martin's idea of professionalism being directly connected to the practice of TDD. Jim repeats this concept at the very end of the GOTO video discussed in this article; he has written[36] contrarian views about TDD in the past, and his sentiments are shared[37] by other prominent bloggers.

(We will not mention Mr. Martin[38] any further in this magazine; it is time to move on[39], and

[25] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=2080s
[26] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=2134s
[27] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=2146s
[28] https://deprogrammaticaipsum.com/write-anywhere-run-once/
[29] https://www.youtube.com/watch?v=ZrBQmIDdls4&t=3016s
[30] https://leanpub.com/ooptheeasyway/
[31] https://www.youtube.com/watch?v=6OQq2kE5wUY
[32] https://byorgey.wordpress.com/2009/01/12/abstraction-intuition-and-the-monad-tutorial-fallacy/
[33] https://kjaer.io/a-monad-is-not-a-burrito/
[34] https://www.youtube.com/watch?v=6OQq2kE5wUY&t=734s
[35] https://www.infoq.com/interviews/coplien-martin-tdd/
[36] https://rbcs-us.com/documents/Why-Most-Unit-Testing-is-Waste.pdf
[37] https://buttondown.email/hillelwayne/archive/i-have-complicated-feelings-about-tdd-8403/
[38] https://www.getrevue.co/profile/tech-bullshit/issues/tech-bullshit-explained-uncle-bob-830918
[39] https://qntm.org/clean

we say this fully aware that this simple phrase will suffice to alienate a substantial segment of our readership. So be it.)

James Coplien's talk "The Dehumanisation of Agile and Objects"[40] is part of the GOTO Conferences YouTube channel[41], by far one of the most interesting YouTube channels you could subscribe to. This particular video includes a very helpful "first comment" from Jim himself, promptly pinned by the GOTO Conference team, with a full list of topics and references to the various papers and books mentioned therein.

Or even better, attend the next GOTO Conference nearby. You will not regret it.

[1]: JAOO is the direct ancestor of the GOTO Conferences. Both conferences were created and managed by the same team at Trifork[42], an organization with which this author has had a fruitful relationship in the past.

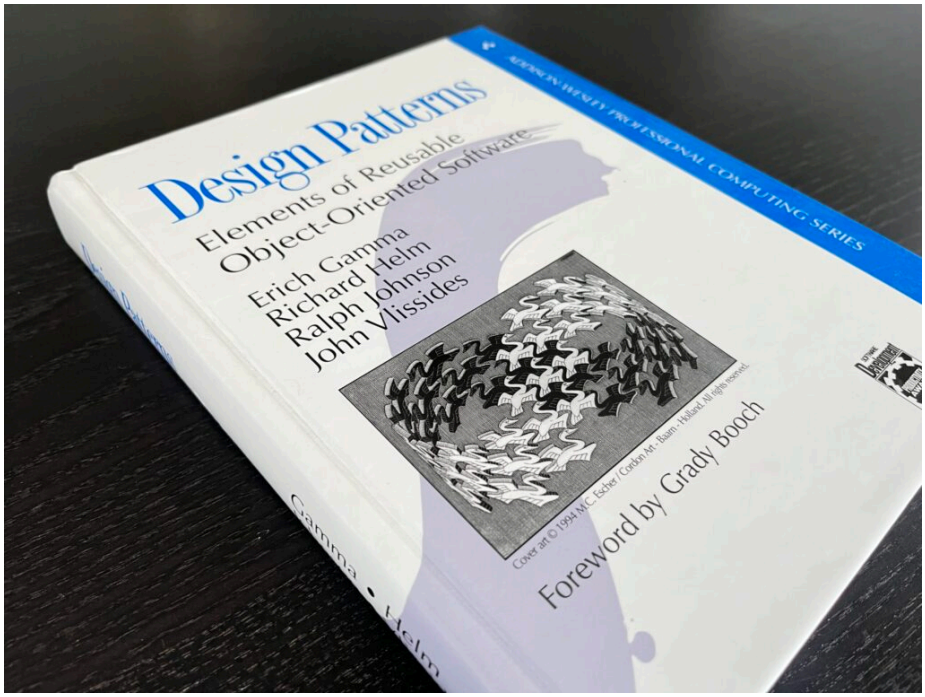
Cover snapshot by the author.

[40] https://www.youtube.com/watch?v=ZrBQmIDdls4
[41] https://www.youtube.com/c/GotoConferences
[42] https://trifork.com/

The Gang Of Four

Adrian Kosmaczewski

October 3rd, 2022



Many different things[1] bear the name "Gang of Four"; however, in this case, we are going to talk about a major bestseller in the history of computer books: "Design Patterns: Elements of Reusable Object-Oriented Software"[2] by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. There is a high probability that every reader of this article owns, has read, or has at least skimmed through the pages of a GoF book once or twice. The book has been reprinted dozens of times (40 times at least until 2012.) It has been the subject of uncountable articles, videos, panel discussions, and, yes, also attacks.

We are not going to delve into the book's contents, structure, or legacy; plenty of commentators have talked about that. Instead, we would like to focus on the various ways the book has been helpful to quite a few generations of computer scientists and software engineers. We will also not dive into the influence of Christopher Alexander's book "A Pattern Language"[3] on the Design Patterns movement. Lots have been said about that. Instead, we will point you to Mr. Alexander's talk at OOPSLA 1996[4].

It is worth a mention (at least) that the GoF book dwarfed another excellent treatise on the subject, also named "Design Patterns for Object-Oriented Software Development,"[5] writ-

[1] https://en.wikipedia.org/wiki/Gang_of_Four_(disambiguation)
[2] https://en.wikipedia.org/wiki/Design_Patterns
[3] https://en.wikipedia.org/wiki/A_Pattern_Language
[4] https://www.youtube.com/watch?v=98LdFA-_zfA
[5] https://archive.org/details/designpatternsfo00pree

ten by Professor Wolfgang Pree[6] and published the same year (1995) by the same publisher (Addison-Wesley) using code examples in the same language (C++.) Pree's book mentions Erich Gamma's 1991 thesis[7] at the University of Zürich as major influence and inspiration.

There are many reasons to think that the GoF book has been influential; after all, even if you do not like Stephen King novels or John Grisham novels, it is undeniable there are a lot of people who enjoyed them. So many, in fact, that Hollywood found on them quite a few successful blockbuster ideas. Yet, at the same time, purists will attack King and Grisham, arguing that they are neither Walt Whitman, Harper Lee, James Joyce, John Steinbeck, or Virginia Wolff. So be it; there is room for many different writers and writing styles.

We should apply the same criteria to computer books. We have already compared Donald Knuth to J. R. R. Tolkien[8]; let us try to find a similar analogy to "Design Patterns."

Structurally speaking, we can read this book in two movements; the first section contains some interestingly advanced (well, at least for 1995) observations on OOP analysis and design. The second section consists of the (in)famous catalog, describing 23 different design patterns. Instead, let us reshuffle the patterns and group them differently.

In our first category, we will place those patterns that proved to be very useful in many contexts: Visitor, Builder, Template Method, Observer, Decorator, Command, Adapter, Façade, and others. In the second group, those that did not: when was the last time you applied the Flyweight, Memento, or Interpreter in your full-stack web application? Finally, there is a small group of eternal outcasts composed of Singleton, Abstract Factory, and Factory Method.

Now for the hard part: these groups are related to Stephen King's novel characters. Yes, you read right. We are pretty sure you did not see that one coming.

Each of the actually helpful OOP patterns in the first group reminds this author of Paul Sheldon, the unfortunate protagonist of "Misery."[9] They are somehow trapped by a group of crazy nurses called Java, C++, and C# (stuck in version 1.0), forcing them to generate lots of classes grouped in curious architectures, all while heavily sedated, unable to escape, and begging for help. At some point, a dynamically typed police force consisting of Ruby, Python, and JavaScript frees them from this ordeal.

The second group is akin to John Smith, the main clairvoyant character of "The Dead Zone."[10] These patterns are constantly aware of a horrendous possible future; they choose to act to prevent it from happening but disappear forever as soon as they jump into action.

Finally, the last group consisting of Singleton and the Factories are like "Carrie,"[11] mocked and bullied by a whole industry in a neverending prom, one which saw them covered with blood and laughed at every coronation ceremony. Yet, they persisted, and every so often, one of them reappears in our code and on our minds while they scream with rage and blow up our unit test suites with all their might in a catastrophic explosion of anger. Well, particularly Singleton, anyway.

Ironically enough, the Gang of Four was first humorously indicted of various sins[12] by their

[6] https://en.wikipedia.org/wiki/Wolfgang_Pree
[7] https://link.springer.com/book/10.1007/978-3-642-77838-4
[8] https://deprogrammaticaipsum.com/how-to-choose-a-programming-language-for-your-book/
[9] https://en.wikipedia.org/wiki/Misery_(novel)
[10] https://en.wikipedia.org/wiki/The_Dead_Zone_(novel)
[11] https://en.wikipedia.org/wiki/Carrie_(novel)
[12] http://www.laputan.org/patterns/trial.html

peers; this happened at OOPSLA 1999, where a "show trial"[13] was held in which the authors were found guilty. One of them, Richard Helm, confessed *in absentia*[14], discharging his responsibility upon the other three accused.

Unfortunately, ten years after the book's publication, John Vlissides passed away[15] at 44, more or less at the same time when "Head First Design Patterns"[16] changed the game of computer books forever.

Despite all the abuse the GoF book has suffered during the past quarter century, it remains a vital and utterly influential piece of work. Sure, we could replace many of these patterns with functional programming constructs, Aspect-Oriented Programming, or what-have-you paradigms[17]. Still, the careful depiction of each pattern in this colossal work as a coherent body of objects interacting at runtime has dramatically changed how we relate to software.

Whether we like it or not, design patterns give names to groups of objects; and Phil Karlton said[18], naming is one of the hardest problems in computer science. We needed this taxonomy, and it proved helpful, if anything, to highlight the infinite flexibility of software and our minds.

Cover photo by the author.

[13] http://www.laputan.org/patterns/gang-of-four.html
[14] http://www.laputan.org/patterns/helm.html
[15] https://www.washingtonpost.com/wp-dyn/content/article/2005/12/09/AR2005120902004.html
[16] https://deprogrammaticaipsum.com/kathy-sierra/
[17] https://deprogrammaticaipsum.com/alan-turing-wrote-object-oriented-code-in-c-and-ran-it-on-beam/
[18] https://skeptics.stackexchange.com/a/39178