



Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



# Java'da Generics



Oğuzhan Çevik · [Follow](#)

Published in Bilişim Hareketi

6 min read · Apr 17, 2020



Share



More

*Yazılım geliştirme sürecinde sorumluluklarımızdan biri de hatalarla uğraşmak ve hataları çözmeye çalışmaktır. Her ne kadar prensiplere uygun kod yazdığımızı ve sağlam bir uygulama geliştirdiğimizi düşünsek de, sürekli gelişen sistemlerde muhakkak bir süre sonra hatalar meydana gelmektedir.*

*Bu tür hataları engellemek için kullandığımız programlama dilleri bizlere bazı özellikler sunmaktadır. Java'da kendi ekosistemindeki geliştiricilere bir çok kolaylık sağlamaktadır. Bunlardan birisi de yazımızın konusu olan Generic tiplerdir.*

## Kısaca Generics Nedir?

Generics, farklı referans veri tiplerini alan, hangi referans tipini alacağına karar verebileceğimiz ve üzerinde benzer işlemler yapabileceğimiz bir sınıftır.



## Generics öncesi

Java 1.4 ve önceki sürümlerde farklı veri tipleri ile çalışma ihtiyacı olduğunda **Object** ve **raw** ile bu ihtiyaç karşılanıyordu.

### // Object ile kullanım

```
List<Object> objectList = new ArrayList<Object>();  
objectList.add("Merhaba Dünya");  
objectList.add(123);  
String string = (String) objectList.get(0);  
Integer integer = (Integer) objectList.get(1);
```

### // Raw ile kullanım

```
List rawList = new ArrayList();  
rawList.add("Merhaba Dünya");  
rawList.add(123);  
String string = (String) rawList.get(0);  
Integer integer = (Integer) rawList.get(1);
```

Listeden farklı türdeki nesnelerimizi çağırmak istediğimizde ise Cast ederek ilgili tipe atama işlemi yapılırdı.

*Tabii Object ve Raw ile kullanım oldukça tehlikeli bir işlemdi. Çünkü fütursuzca kullanımda çalışma zamanında RuntimeException almamız kaçınılmazdı.*

Aşağıdaki kodda çalışma zamanında **ClassCastException** hatasını alırız.

```
List list = new ArrayList();  
list.add("Merhaba Dünya");
```

```
list.add(123);  
for (int i = 0; i < list.size(); i++) {  
    String string = (String) list.get(i);  
}
```

## Generic Sonrası

2004'de Java 5 çıktığında beraberinde çok önemli özellikler geldi. Autoboxing Unboxing, Foreach, Static Import, Concurrency iyileştirmeleri vs.

Bu önemli özelliklerin yanında birde Generics ifadeler dile kazandırıldı. Generics ifadelerin gelmesiyle dil büyük bir gelişim ve değişime uğradı. Birçok java kütüphanesi Generics ifadeler ile yeniden yazıldı.

## Nasıl kullanılır?

Generics ifadeler Interface, Class, Metohod'larda kullanılabilir.

```
// Interface ile kullanım  
public interface Comparable<T> {  
    public int compareTo(T object);  
}  
  
// Class ile kullanım  
public class Printer<T> {  
    public void print(T object) {  
        System.out.println(object);  
    }  
}  
  
// Method ile kullanım -1  
public <T> T myGenericMethod(){  
    T object = null;  
    /**/  
    return object;  
}  
  
// Method ile kullanım -2  
public <T> T myGenericMethodWithParameter(T object){  
    return object;  
}  
  
// Method ile kullanım -3  
public <T> void myGenericVoidMethod(T object){  
    System.out.println(object);  
}
```

## Generics Avantajları, Neden kullanmalıyız?

## Why we use generics? Because Generics stop runtime errors at compile time.

- Genericler uygulamalarınızda ki hataları minimuma indirmeyi amaçlar. Özellikle Runtime sırasında oluşacak ClassCastException hatalarını derleme sırasında yakalayabiliriz.

Yukarıda ClassCastException hatasını aldığımız kodu hatırlayalım. Ve oradaki kod bloğunu Generic ifadeler kullanarak yeniden yazalım.

```
// Generic ifadeler kullanılarak
List<String> list = new ArrayList<String>();
list.add("Merhaba Dünya");
list.add(123);
for (int i = 0; i < list.size(); i++) {
    String string = list.get(i);
}
```

Yukarıdaki kod derlenmez. Çünkü listemizin sadece String tipindeki verileri kabul edeceğini bildiriyoruz. Dolayısıyla çalışma zamanında alacağımız hatayı derleme sırasında alırız ve derleyici bizi uyarıp hatayı düzeltmemizi ister.

*Java tip ihlaline dikkat eden bir dildir. Derleme zamanındaki hataları bulmak çalışma zamanındaki hataları bulmaktan daha önemlidir, daha doğru bir yaklaşımdır. Bu nedenle Generics ifadelerle yazılan kodlar derleme zamanında işleme alınır ve hatayı yakalamak çok daha kolaydır.*

*Ayrıca Generic ifadeleri kullanarak uygulamamızdaki kodları **type-safety** olarak yazarız.*

**Type Safety:** Tip güvenli, tip doğrulaması anlamına gelmektedir. Java, C# gibi diller type-safety dillerdir. Yani bir değişken tanımlamak istediğimizde hangi tipte bir değişken olduğunu bildirmek zorundayız. Derleyici derleme sırasında tüm değişkenler için doğrulama yapar ve yanlış bir atama yapıldığında hata verir.

- Genericler, referans veri tiplerini tek bir yapı altında toplayarak kod tekrarını engellemiş olur ve daha temiz kod yazmamızı sağlar. Ayrıca uygulamalarımızda kullandığımız kodların çok daha etkili bir şekilde tasarlanmasına yardımcı olur.

```
// Generic kullanmadan
PersonRepository {
    savePerson(Person person) {
```

```
        ***
    }
}

AnimalRepository {
    saveAnimal(Animal animal) {
        ***
    }
}
```

Yukarıdaki kod örneğinde görüldüğü üzere iki farklı tipdeki veri veritabanına kaydedilmek isteniyor. Generic kullanmadan iki ayrı method yazılmış ve aslında ikiside aynı işi farklı tipler için yapıyor.

```
// Generic kullanarak
GenericRepository <T> {
    save(T object) {
        ***
    }
}
```

Yukarıda Generic bir tanımlama yaparak önceki iki method ile aynı görevi yapan tek bir method yazarak kod fazlalığından kurtulmuş olduk.

- Generics ifadelerin bir diğer faydası Casting işlemine gerek duymamasıdır. Java 1.4 ve önceki versiyonlarda Object veya Raw ile kullanımda Casting işlemi zorunluydu ve gereksiz kod kalabalığı vardı.

```
// Java 5 Öncesi
List list = new ArrayList();
list.add("Merhaba");
String str = (String) list.get(0);

// Java 5 Sonrası
List<String> genericList = new ArrayList<String>();
genericList.add("Merhaba");
str = genericList.get(0);
```

## Java 7 ile Generic instance oluşturma

Java 7'de derleyici daha akıllı hale getirildi. Generic instance oluşturduğumuzda sağ tarafta generic tipi tekrar belirtme zorunluluğu kaldırıldı.

```
// Java 7 öncesi
```

```
List<String> list = new ArrayList<String>();
```

```
// Java 7 sonrası
```

```
List<String> list = new ArrayList<>();
```

### Neden Generic tip tanımlarken T kullanılıyor? Neden X,Y,Z değil?

Generic ifadelerde dikkat etmişsinizdir hep T harifini kullandık. Bunun nedeni Java Naming Convention (Java İsimlendirme Standartları).

Harf	Anlamı	Açıklama
E	Element	Genelde Java Collection Framework de kullanılmaktadır
K	Key	
N	Number	
T	Type	
V	Value	
S,U,V vb.		2., 3., 4. tipler gerekirse de bu harfler seçiliyor

Open in app ↗



Search



```
V put(K key, V value);  
}
```

### Bounded Types

Generic ifadeler kodlarımızda birden fazla veri tipi ile çalışmamıza olanak sağlar. Ama bazen uygulamamızda öyle ihtiyaçlarımız oluyor ki sadece tek bir türden nesne ile çalışmak isteyebiliriz.

Örneğin karenin alanını hesaplayacak olursak buradaki veri tipimiz numeric tipler olacaktır. Integer, Double, Long vs. Bu sınıfların hepsi **java.lang.Number** sınıfını extend ediyorlar.

**Bounded types**, oluşturulacak Generic yapıyı belirli bir şablon altında sınırlandırmamızı sağlar. Kısaca Generic yapımız belirli bir sınıftan türesin istiyorsak bounded type ile belirtmemiz gerekiyor.

```
public class Kare<T extends Number> {  
  
    public double alanHesapla(T uzunluk) {  
        return uzunluk.doubleValue() * uzunluk.doubleValue();  
    }  
  
}
```

## Multiple Bounds

İstedüğimiz kadar sınırlandırma yapabiliriz.

```
public static <T extends Object & Comparable<? super T>> T  
max(Collection<? extends T> coll) {...}
```

## Wildcard'lar

Wildcard ifadeler önceden T,K,V gibi isimlendirmelerin yerine ? ile ifade edilir. Bilinmeyen tip anlamına gelir. Wildcard'lar 3'e ayrılır. Unbounded, Upper Bounded ve Lower Bounded.

### Unbounded Wildcard

Kalıtım ile sınırlandırılmamış wildcard anlamına gelir. Tipini bilmediğimiz verileri okumak için kullanılır.

```
public void printList(List<?> list) {  
    for (Object object : list) {  
        System.out.println(object);  
    }  
}
```

### Bounded Wildcard

Kalıtım ile sınırlandırılmış olan wildcard anlamına gelir. Kendi içinde ikiye ayrılır. Upper Bounded ve Lower Bounded.

### Upper Bounded Wildcard

Kalıtım yapısını **extends** anahtar kelimesi ile kullanır. Listedeki okuma yapıldığı zaman kullanırız.

```
public void read(List<? extends Number> list){  
    for (int i = 0; i < list.size(); i++){  
        System.out.println(i);  
    }  
}
```

```
}  
}
```

## Lower Bounded Wildcard

Kalıtım yapısını **super** anahtar kelimesi ile kullanır. Listeye ekleme yapıldığı zaman kullanırız.

```
public void addNumbers(List<? super Number> list) {  
    for (int i = 1; i <= 10; i++) {  
        list.add(i);  
    }  
}
```

## Type Erasure

*Type erasure is the technique using which the Java compiler translates generic parameterized type to raw type in Java generics.*

Derleme sırasında tanımladığımız sınıf ve nesnelerin tip bilgilerinin silinmesine denir. Derleyici, kodumuzda yazdığımız Generic ifadeleri anlamakla sorumludur. Dolayısıyla derleyici bizim yazmış olduğumuz Generic ifadeleri kendi anlayacağı dile çevirir.

## Method Erasure

```
// Derlemeden Önce  
public static <E> void printArray(E[] array) {  
    for (E element : array) {  
        System.out.println(element);  
    }  
}  
  
// Derlendikten Sonra  
public static void printArray(Object[] array) {  
    for (Object element : array) {  
        System.out.println(element);  
    }  
}
```

## Class Erasure



**// Derlemeden Önce**

```
public class Stack<E> {
    private E[] stackContent;

    public Stack(int capacity) {
        this.stackContent = (E[]) new Object[capacity];
    }

    public void push(E data) {
        // ..
    }

    public E pop() {
        // ..
    }
}
```

**// Derlendikten Sonra**

```
public class Stack {
    private Object[] stackContent;

    public Stack(int capacity) {
        this.stackContent = (Object[]) new Object[capacity];
    }

    public void push(Object data) {
        // ..
    }

    public Object pop() {
        // ..
    }
}
```

Derlemeden Önce	Derlendikten Sonra
List<String>	List
List<Integer>	List
List<List<Integer>>	List
List<String>[]	List[]
T	Object
T extends Number	Number

**Type Erasure ve Overloading**

Buraya kadar Type Erasure kavramını anladık. Bunu Overloading örneği ile doğrulayalım.

```
public void print(String param){}
public void print(Integer param){}
public void print(List param){}
```

Buraya kadar kodumuz derlenecektir. Ama aşağıdaki kod eklenince kodumuz derlenmeyecektir.

```
public void print(List<String> param);
```

Derlenmemesinin sebebi tabiki **Type Erasure**. Derleyici son iki methodun da aynı method olduğunu kabul eder.

Bu yazıda genel olarak Java'da Generic ifadelerden bahsetmeye çalıştım. Umarım faydalı bir yazı olmuştur.

Java

Generics

Wildcard

Erasure

Yazılım



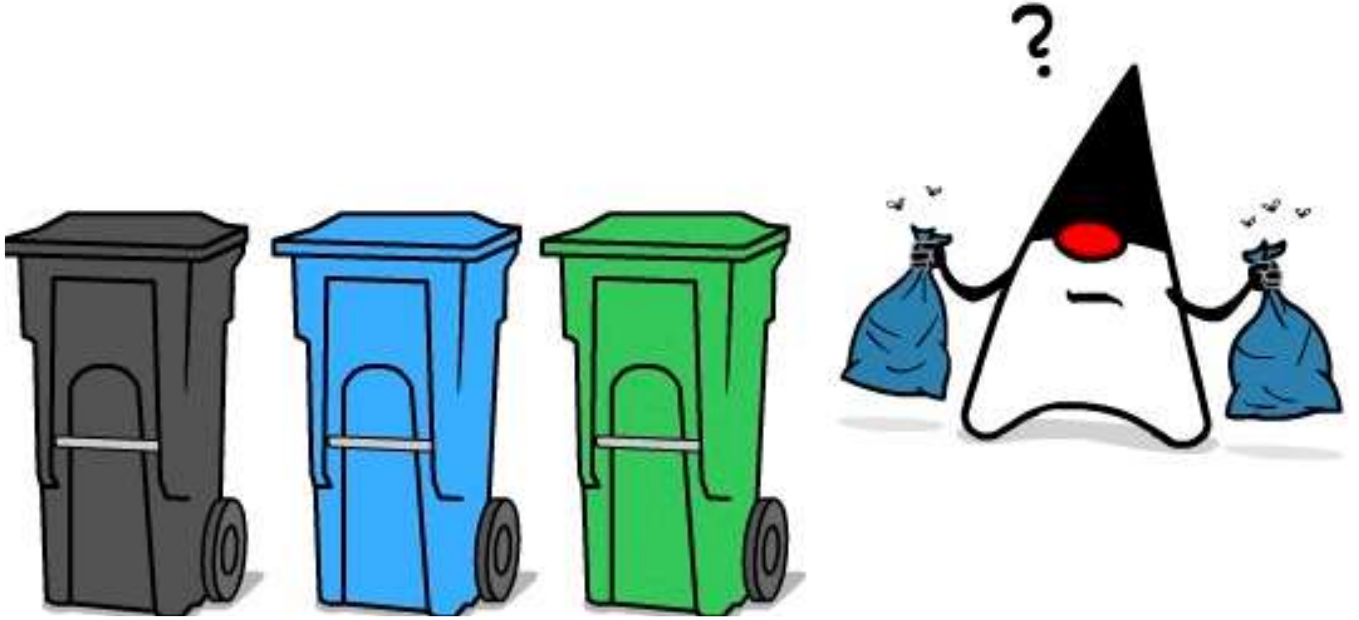
Follow

**Written by Oğuzhan Çevik**

233 Followers · Writer for Bilişim Hareketi

Software Developer

More from Oğuzhan Çevik and Bilişim Hareketi



Oğuzhan Çevik in Bilişim Hareketi

## Java'da Bellek Yönetimi

Hepimiz hemen her gün onlarca, hatta yüzlerce satır kod yazıyoruz ama çoğu kez işin arka tarafında mutfakta bu işlerin nasıl yürütüldüğünü...

6 min read · Jul 16, 2019



212



2





## Veri Bilimi İçin Temel Python Kütüphaneleri-1 : Numpy

NumPy (Numerical Python) bilimsel hesaplamaları hızlı bir şekilde yapmamızı sağlayan bir matematik kütüphanesidir. Numpy'ın temelini numpy...

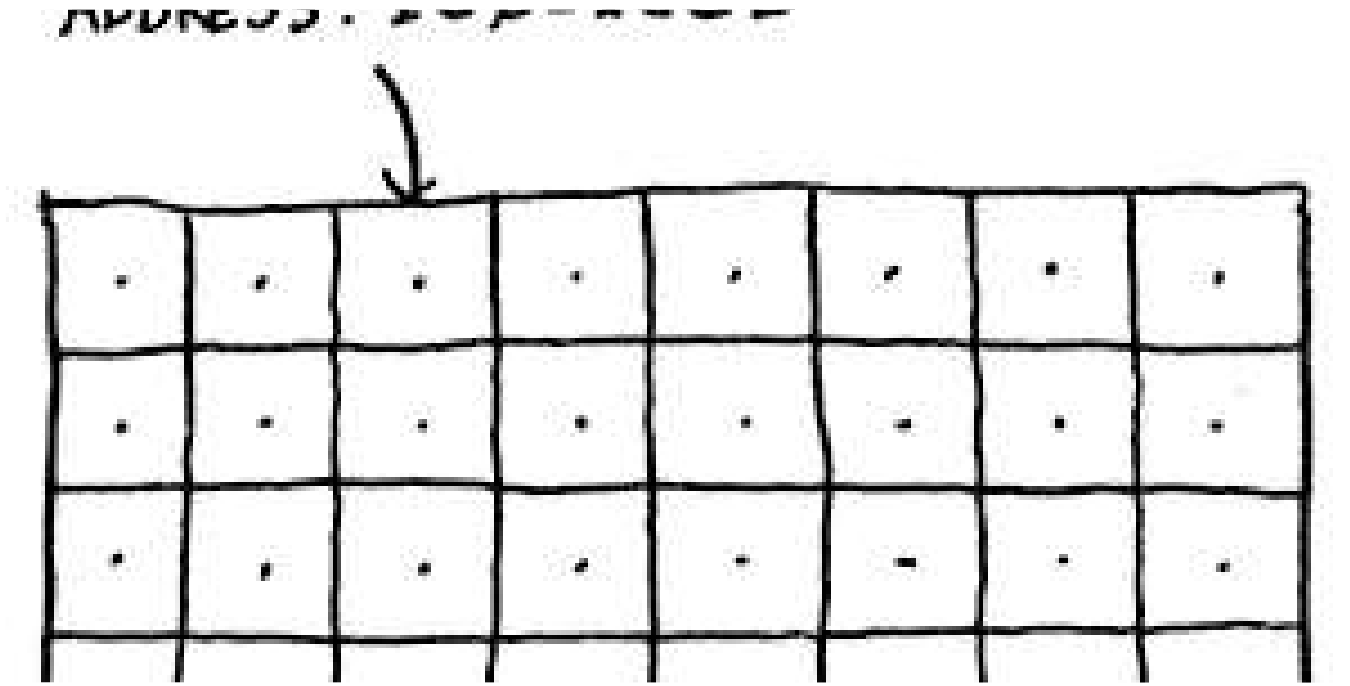
7 min read · Feb 1, 2019



530



2



Oğuzhan Çevik in Bilişim Hareketi

### Dizi mi? Liste mi? Hangisini kullanmalıyım?

Merhaba, bu yazı Grokking Algorithms kitabındaki bazı bölümlerin çevirisidir. Bu yazıyı yazmama izin veren kitabın yazarı Aditya...

6 min read · Nov 7, 2020



108



2



See all from Oğuzhan Çevik

See all from Bilişim Hareketi

## Recommended from Medium

```
private Mono<ServerResponse> updateOnePersonById(ServerRequest request) {  
    return request.bodyToMono(String.class).  
        flatMap(this::readPersonFromRequestBody).  
        filter((person) ->  
            person.getId().equals(  
                request.pathVariable(PERSON_ID_PATH_VARIABLE)  
            )  
        ).  
        flatMap(this.personService::updateOneById).  
        flatMap((success) -> success ?  
            Responses.noContent() : Responses.internalServerError()  
        ).  
        switchIfEmpty(Responses.badRequest()).  
        subscribeOn(Config.APPLICATION_SCHEDULER);  
}
```



Marcus Höjvall in Alphadev thoughts

## The Reactive Java era is over. Here is why.

It was actually over a few years ago but with the newly released Virtual threads in Java 21, many people now realize their mistake.

🌟 · 4 min read · Feb 20, 2024

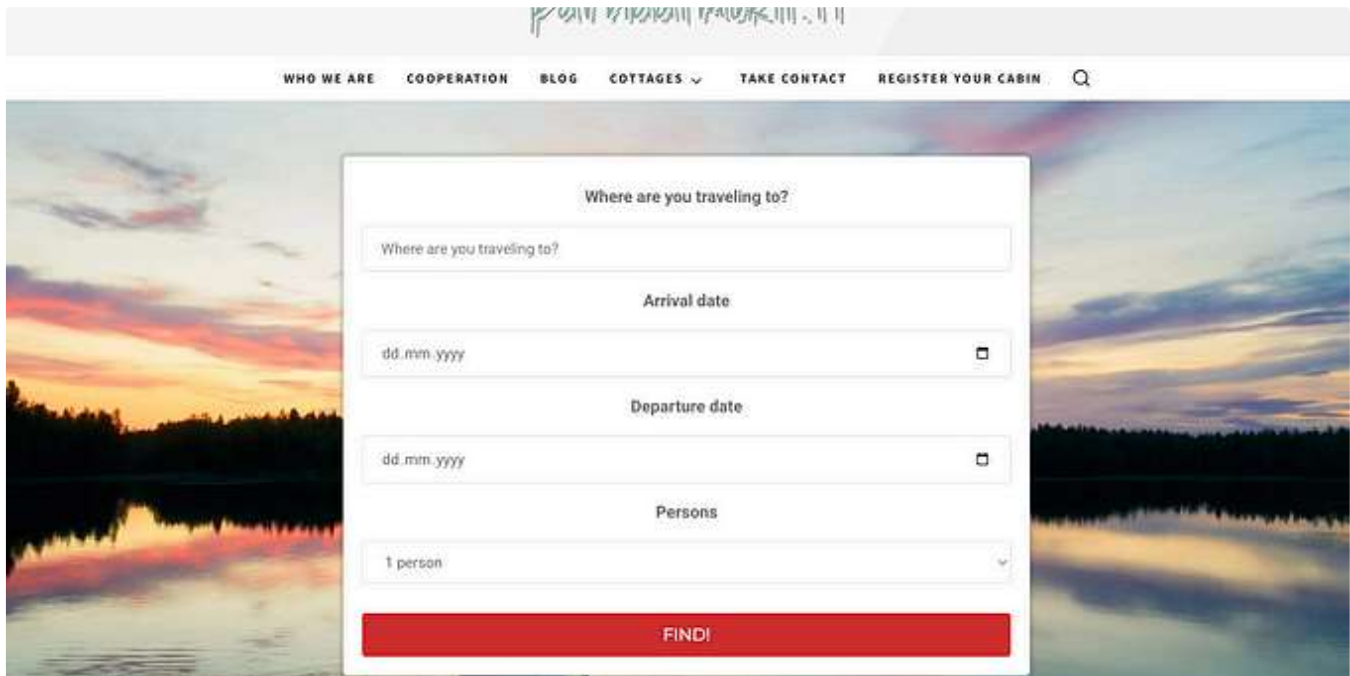



974



22





 Artturi Jalli

## I Built an App in 6 Hours that Makes \$1,500/Mo

Copy my strategy!

🌟 • 3 min read • Jan 23, 2024

 12.4K    146

### Lists



#### General Coding Knowledge

20 stories • 992 saves




#### data science and AI

40 stories • 94 saves





 Vaishnav Manoj in DataX Journal

## JSON is incredibly slow: Here's What's Faster!

Unlocking the Need for Speed: Optimizing JSON Performance for Lightning-Fast Apps and Finding Alternatives to it!

16 min read · Sep 28, 2023



13.9K



163



Karolina Kozmana



## Common side effects of not drinking

By rejecting alcohol, you reject something very human, an extra limb that we have collectively grown to deal with reality and with each...

10 min read · Jan 22, 2024



22K



616



Alexandru Lazar in ILLUMINATION

## Ten Habits that will get you ahead of 99% of People

Improve your life and get ahead of your peers in 10 simple steps

9 min read · Nov 18, 2023



20K



367



Companies have found that "Agile", as it is sold, delivered, and explained to them, does not work. You can blame them if you like, or you can blame the Agile community for not packaging the right kinds of learning and support. But regardless, "Agile" as we know it is dead. And Scrum will go with it.

But companies still need `_agility_`. Real agility. That has been our focus.

Real agility is mostly behavioral, and in particular, it is driven by the behaviors of leaders. Leadership is the big glaring hole in the Agile Manifesto. It is like trying to make concrete without water. No wonder "Agile" did not work.

That's why Agile 2, which reimagined what "Agile" should have been,



Tamás Polgár in Developer rants

## Agile has failed. Officially.

Either I'm a gifted oracle, and all of my friends are, or Agile really was just a stupid idea to begin with. After many years of agony...

2 min read · Dec 2, 2023



15.3K



478



See more recommendations