NESNEYE YÖNELİK YAZILIM MÜHENDİSLİĞİ YIL SONU ÖDEVİ

SEZERİN KÜÇÜK

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ

21743393

ANLATILAN KONULAR

Command Pattern

Vendor Lock-In

Boat Anchor

Corncobs

COMMAND PATTERN

Bir isteği bir nesne olarak saklar, böylece istemcileri farklı isteklere, sıraya veya günlük isteklerine göre parametrelendirmenize ve geri alınamayan işlemleri desteklemenize izin verir. Davranışsal tasarım kalıplarından biri olan komut tasarım kalıbı, nesneye dayalı programlama (Object Oriented Programming) çalışmalarında çoğu zaman farkında olmadan kullanılan bir yapıdır. Komut tasarım kalıbı, kullanıcı isteklerini gerçekleştiren kod yapısının sarmalanarak nesneler halinde saklanmasına dayanır. Öyle ki üzerinde çalışılacak nesnenin tanımının yapılamadığı durumlar olabilir. Bu şartlarda ne tür çözüm yolları ile nesneye müdahale edilmeye çalışılabileceği kestirilemez, ancak gerçekleştirilmek istenen işlemler bir nesne olarak sarmalanır. Nesne halinde tutulan bu sarmal kod yapısı, alıcı nesne için bir çözüm oluşturur. Çözümlerin nesneler halinde saklanmasının getirisi olarak da komut tasarım kalıbı aynı kod yapısının tekrar tekrar kullanılabilmesine olanak sağlar.

Komut tasarım kalıbının kullanıldığı tüm durumlarda geçerli olan bazı ortak terimler mevcuttur. Açıklamaları ile beraber bunlar;

Komut (Command): Gerçekleştirilecek işlem için bir ara yüz tanımlar.

Somut Komut (Concrete Command): Alıcı ve gerçekleştirilecek işlemler arasında bir bağ kurar, alıcıda karşılık düşen işlemleri çağırarak çalıştırma eylemini gerçekleştirir.

İstemci (Client): Komut nesnesini oluşturur ve metodun sonraki zamanlarda çağrılabilmesi için gerekli bilgiyi sağlar.

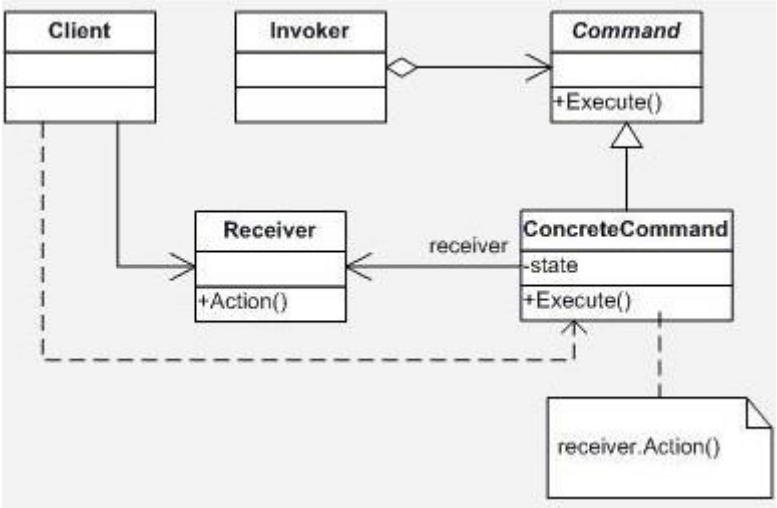
Çağırıcı (Invoker): Metodun ne zaman çağrılacağını belirtir.

Alıcı (Receiver): Kullanıcı isteklerini gerçekleştirecek asıl metod kodlarını içerir.

Komut, işlemi nasıl gerçekleştireceğini bilen birinden işlemi başlatan nesneyi ayırır. Bu ayrımı başarmak için tasarımcı, bir alıcıyı (nesne) bir eylemle (üye işlevine bir işaretçi) eşleyen soyut bir temel sınıf oluşturur. Temel sınıf, sadece alıcıdaki eylemi çağıran bir execute () yöntemini içerir.
Komut nesnelerinin tüm istemcileri, nesnenin istemcinin "hizmetini" istediği zaman nesnenin sanal yürütme () yöntemini çağırmak suretiyle her nesneyi "kara kutu" olarak ele alır.
Bir komut sınıfı, aşağıdakilerin bazı alt kümelerini içerir: bir nesne, nesneye uygulanacak yöntem ve yöntem uygulandığında geçirilecek bağımsız değişkenler. Komutun "yürütme" yöntemi daha sonra parçaların bir araya gelmesine neden olur.
Komut nesnelerinin dizileri, bileşik (veya makro) komutlarına birleştirilebilir.

Bir komut oluşturan istemci, onu yürüten aynı istemci değildir. Bu ayrım, komutların zamanlaması ve sıralamasında esneklik sağlar. Komutları nesneler olarak biçimlendirmek, bir masaya geçirilebilmeleri, sahneye konabilmeleri, paylaşılabilmeleri, yüklenebilmeleri ve başka herhangi bir nesne gibi başka araçlarla çalınabilmeleri veya manipüle edilebilmeleri anlamına gelir.

Komutların nesne halinde tutulmasının getirdiği bir fayda olarak istenildiği durumlarda bu nesnelere yaptıkları işlemleri geri almak da öğretilebilir. Bunun sonucu olarak **Yinele(Redo)/Geri Al(Undo)** işlemleri gerçeklenebilir ve bir makro özelliği yaratılabilir.



Şekil 1. Komut tasarım kalıbı için UML diyagramı

Yukarıdaki Şekil 1'de Command Pattern UML diyagramıdır. Şekilde de görüldüğü gibi tüm komutlarımız (Command) commaand base sınıfı altında tek bir ara yüzde toplanmıştır. Bu bize kontrol panelindeki butonlara Command tipinde nesneler atamamızı sağlamaktadır. Yapılmasını istediğimiz her bir istek için yeni bir Command sınıfı oluşturmamız gerekmektedir.

Komut nesneleri, ne yapılması gerektiğini bilen ve müşterinin bunu yapması için gerekli kaynakları olan başka bir müşteriye iletilen bir müşteri tarafından oluşturulan "belirteçleri" olarak düşünülebilir.

Komut Tasarım Kalıbı İle İlgili Uygulama:

Verilen uygulamada akış sırası kod içerisinde özellikleri belirtilmiş Kullanici sınıfından kullanici adında bir nesne oluşturulması ile başlar. Kullanici sınıfı içerisinde bu sınıftan türetilmiş her nesne için yine özellikleri kod içerisinde belirtilmiş Hesap sınıfından hesap adında bir diğer nesne oluşturulur. Hesap sınıfı ise mevcut değişken ve belirtilen değişken arasında belirtilen operatöre karşılık gelen işlemin gerçekleştirildiği kısımdır. Gerçekleştirilen her işlem bir seviye belirtir ve her seviyede çalışan komut, komut listesine yeni bir eleman olarak eklenir. Yapılan işlemleri geri almak için mevcut değişken üzerinde o ana kadar gerçekleşmiş olan işlemler komut listesinden alınır, GeriAl fonksiyonu ile tersi işlemler oluşturulur ve istenen seviye kadar geri dönülür. Yapılan işlemleri yinelemek için ise yine aynı düşünce ile o ana kadar gerçekleşmiş olan işlemler komut listesinden alınır, IleriAl fonksiyonu ile istenen seviye kadar yinelenir.

```
using System;
using System.Collections.Generic;
namespace KomutKalibi
{
   class MainApp

   {
      static void Main()

      {
         Kullanici kullanici = new Kullanici();
         kullanici.Hesapla('+', 100);
         kullanici.Hesapla('-', 50);
         kullanici.Hesapla('*', 10);
```

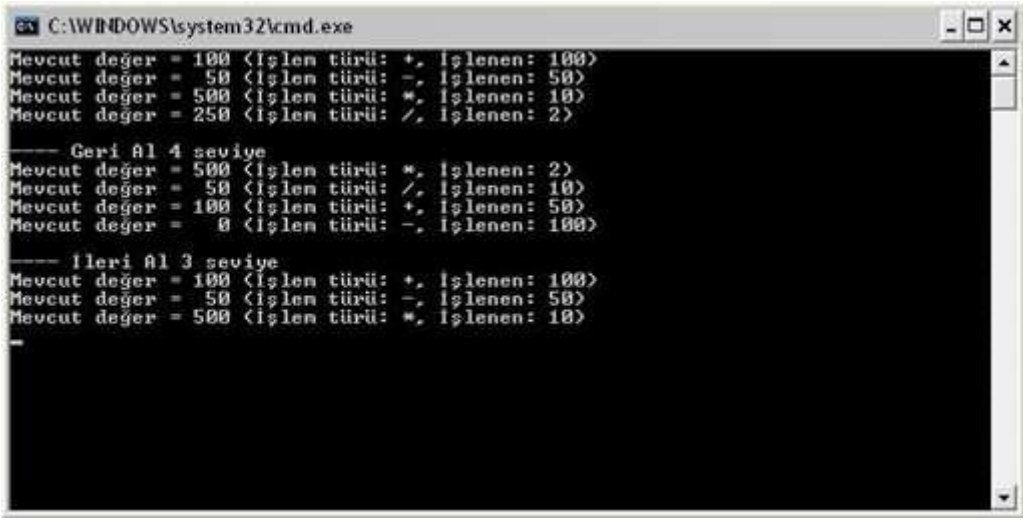
```
        kullanici.Hesapla('/', 2);
        kullanici.GeriAl(4);
        kullanici.IleriAl(3);
        Console.ReadKey();
    }
}
abstract class Komut
{
    public abstract void Uygula();
    public abstract void Uygulama();
}
class HesapKomutu : Komut
{
    private char _islemTuru;
    private int _islenen;
    private Hesap _hesap;
    public HesapKomutu(Hesap hesap, char @islemTuru, int islenen)
    {
        this._hesap = hesap;
        this._islemTuru = @islemTuru;
        this._islenen = islenen;
    }
    public char IslemTuru
    {
        set { _islemTuru = value; }
    }
    public int Islenen
    {
        set { _islenen = value; }
    }
    public override void Uygula()
    {
        _hesap.Islem(_islemTuru, _islenen);
    }
    public override void Uygulama()
    {
        _hesap.Islem(GeriAl(_islemTuru), _islenen);
    }
    private char GeriAl(char @islemTuru)
    {
        switch (@islemTuru)
        {
            case '+': return '-';
            case '-': return '+';
            case '*': return '/';
            case '/': return '*';
            default: throw new
             ArgumentException("@islemTuru");
        }
    }
}
```

```
   class Hesap
   {
      private int _mevcut = 0;
      public void Islem(char @islemTuru, int islenen)
      {
         switch (@islemTuru)
         {
            case '+': _mevcut += islenen; break;
            case '-': _mevcut -= islenen; break;
            case '*': _mevcut *= islenen; break;
            case '/': _mevcut /= islenen; break;
         }
         Console.WriteLine("Mevcut değer = {0,3} (İşlem türü: {1}, İşlenen: {2})", _mevcut, @islemTuru,
islenen);

      }
}
   class Kullanici
   {
      private Hesap _hesap = new Hesap();
      private List<Komut> _komutlar = new List<Komut>();
      private int _mevcut = 0;
      public void IleriAl(int seviye)
      {
         Console.WriteLine("\n---- İleri Al {0} seviye ", seviye);
         for (int i = 0; i < seviye; i++)
         {
            if (_mevcut < _komutlar.Count - 1)
            {
               Komut komut = _komutlar[_mevcut++];
               komut.Uygula();
            }
         }
      }
      public void GeriAl(int seviye)
      {
         Console.WriteLine("\n---- Geri Al {0} seviye ", seviye);
         for (int i = 0; i < seviye; i++)
         {
            if (_mevcut > 0)
            {
               Komut komut = _komutlar[--_mevcut] as Komut;
               komut.Uygulama();
            }
         }
      }
      public void Hesapla(char @islemTuru, int islenen)
      {
         Komut komut = new HesapKomutu(_hesap, @islemTuru, islenen);
         komut.Uygula();
         _komutlar.Add(komut);
```

```
            _mevcut++;
        }
    }
}
```

Uygulamanın ekran çıktısı aşağıdaki gibidir:



ANTI PATTERNS

1.VENDOR LOCK-IN

Satıcı kilidi, bir ürünü veya hizmeti kullanan bir müşterinin bir rakibin ürününe veya hizmetine kolayca geçemediği bir durumdur. Satıcı kilidi, genellikle rakiplerinkiyle uyumsuz olan özel teknolojilerin sonucudur. Bununla birlikte, başka şeylerin yanı sıra, verimsiz süreçler veya sözleşme kısıtlamalarından da kaynaklanabilir. Satıcı kilitlenmesinden duyulan korku genellikle bulut hizmetinin benimsenmesi için büyük bir engel olarak gösteriliyor. Bulut hizmeti geçişinin karmaşıklığı, pek çok müşterinin hantal bir süreçten kaçınmak için ihtiyaçlarını karşılamayan bir sağlayıcıyla kalması anlamına gelir. Verileri bir sağlayıcı'nın bulut ortamından diğerine taşımak için, örneğin, verileri önce müşterinin sitesine geri taşımak ve ardından yeni sağlayıcının ortamına taşımak genellikle gereklidir. Ayrıca, veriler orijinal tedarikçinin sistemiyle uyumluluk için değiştirilmiş olabilir, böylece müşteriye iade edilen şey tekrar hareket ettirilmeden önce eski durumuna geri gönderilmelidir.

Satıcı kilitlemesinden kaçınmanın en iyi yolu, hizmetinizi en başta akıllıca seçmektir. Depolama uzmanı Arun Taneja, bulut satıcısının kilitlenmesini önlemek için aşağıdaki ipuçlarını sunmaktadır:
Her bir sağlayıcının politikalarının ince baskısını okuyun ve gerekiyorsa, onlara doğrudan hareketli müşteri verilerini bulut depolama havuzlarından nasıl çıkardıklarını sorun.
Sağlayıcıdan, büyük miktarlardaki verilerin hareketini kolaylaştırmak için veri taşıma araçları veya hizmetleri sunup sunmadıklarını sorun.
Depolama Ağı Endüstrisi Birliği (SNIA) tarafından oluşturulan Bulut Veri Yönetimi Arayüzü (CDMI) standardı gibi, gelişmekte olan endüstri standartlarını desteklemek için söz vermiş sağlayıcıları seçin.

Satıcı kilidi, satıcı hizmetlerine müşteri bağımlılığı sağlayan bir hizmet sunum tekniğidir. Bu, özel yazılım / uygulama / donanım / ekipmana göre platform bağımlı olan ve sınırlı ve üçüncü taraf satıcı ortaklarıyla münhasıran veya birlikte çalışan BT çözümleri geliştirerek başarılır. Üstelik, bu tür hizmetler, rakip satıcılar arasında yüksek anahtarlama maliyetlerini ortadan kaldırmakta, müşterileri isteksiz hale getirmekte ya da hatta farklı satıcılara geçiş yapamamaktadır. Bu, iyi teknoloji ve iyi iş arasındaki bir

tradeoff vakası. Ne yazık ki müşteriler (ve teknisyenler) için, kötü teknoloji ve kapalı standartlar bazen daha karlı bir ürün üretmektedir.
Bir örnek, bir SIM kartın belirli bir telefon üreticisine ait olduğu bir SIM kilidi (ağ kilidi). Microsoft, genellikle açık kaynaklı uygulama programlama arabirimleri (API) ve ağır işletim sistemi anahtarlama maliyetleri gerektiren Windows işletim sistemi (OS) uygulamaları için üreticinin kilitlenmesi için satıcı tarafından eleştirilmektedir. Satıcı kilitleme, bulut bilişim çözümlerinde belirgin bir şekilde görünür durumdadır; veri / hizmet değişimi ve entegrasyon, her zaman maliyetli olmasa da, oldukça karmaşıktır.

Satıcı kilitlenmesini önlemek için atılacak adımlar
1) durum tespiti yapmak
2) Çıkış için erken plan yapın
3) Uygulamanızı gevşek bir şekilde birleştirin.
4) Verilerinizin taşınabilirliğini en üst düzeye çıkarın
5) Çok bulutlu bir strateji düşünün
6) DevOps araçlarını ve süreçlerini uygulamak

Sonuç
Uygulama mantığınız ve verileriniz tek bir bulut servis sağlayıcısı tarafından tutulduğunda ortaya çıkabilecek sorunlar var.
Bulut satıcısının kilitlenmesini hafifletmek için, potansiyel bulut sağlayıcılarına gereken özeni göstermeli, bir çıkış için erken plan yapmalı, gevşek birleştirilmiş bir uygulama oluşturmalı, verilerinizi maksimum taşınabilirlik için ayarlamalı, çok bulutlu bir strateji düşünmeli ve DevOps araçlarını uygular.
Bulut bilişim satıcısı kilitleme yasal bir endişe olsa da, faydaların risklere ağır bastığına inanıyoruz ve CSP'nizi derinden ele alarak çok daha fazla değer elde edip daha iyi uygulamalar oluşturabiliyorsunuz.

2.BOAT ANCHOR

Bir Tekne Çapası, mevcut proje üzerinde yararlı bir amaç taşımayan bir yazılım veya donanım parçasıdır. Genellikle, Tekne Çapa, satın alma işlemini daha da ironik hale getiren pahalı bir satın almadır.
Bir Tekne Çapanın elde edilmesinin nedenleri genellikle o zaman zorlayıcıdır. Örneğin, bir politika veya programatik ilişki, belirli bir donanım veya yazılım parçasının satın alınmasını ve kullanılmasını gerektirebilir. Bu, yazılım projesinin bir başlangıç varsayımıdır (veya kısıtlamasıdır). Bir başka zorlayıcı sebep ise, bir kilit yöneticinin satın alma işleminin faydası olduğuna ikna edilmesidir.
"Çok önemli bir kişi (VIP) pazarlaması" adı verilen bir satış uygulaması, satın alma yetkisine sahip olan üst düzey karar vericilerin satışlarını hedefler. VIP pazarlama genellikle küçük ve orta ölçekli şirketlerden sorumlu icra görevlilerine odaklanır. Ürüne uygun bir teknik değerlendirme olmaksızın bir taahhüt verilir.

Yöneticiler ve yazılım geliştiricilerinin sonuçları, ürünün işe yaraması için önemli çaba sarf edilmesi gerektiğidir.
Önemli bir zaman ve kaynak yatırımından sonra, teknik personel, ürünün mevcut bağlamda işe yaramadığını fark eder ve bunu başka bir teknik yaklaşım için terk eder. Sonunda, Tekne Ankrajı bir kenara bırakılır ve bazı köşelerde (donanım varsa) tozu toplar.

Amatör radyo ve bilgi işlemlerde , bir tekne çapası veya boaboğazı , eskimiş, işe yaramaz ve hantal bir şeydir - metaforik olarak adlandırıldığı için tek üretken kullanımı bir tekne demirleme olarak suya atılmaktı

Refactored Çözüm
İyi mühendislik uygulamaları, teknik yedekleme için gerekli olan en az yazılım yeniden işlenmesiyle kurulabilecek alternatif bir yaklaşımı içerir. Teknik yedekleme seçimi önemli bir risk azaltma stratejisidir.
Çoğu altyapı teknolojisinde (çoğu yazılımın bağlı olduğu) ve yüksek risk altındaki diğer teknolojiler için teknik yedeklemeler tanımlanmalıdır. Teknik yedeklemeler seçim sürecinde kritik yol teknolojileriyle birlikte değerlendirilmelidir. Hem kritik yol hem de yedekleme teknolojileri için değerlendirme lisansları ile prototipleme (çoğu satıcıdan edinilebilir) önerilir.

Tekne çapası terimi, genellikle daha sonra gerekli olması durumunda, bir sistemin kod tabanı içinde bırakılan yazılım koduna [6] uzatılmıştır.

Bu bir anti-desen örneğidir ve bu nedenle eski kodu içeren programı sürdürmeye çalışan insanlar için birçok soruna neden olabilir. En önemli sorun, programcıların, hiçbir şey yapmayan ve çalışma kodu olmayan eski kodları birbirinden ayıran zor bir zamana sahip olmalarından kaynaklanmaktadır. Örneğin, bir programcı programın giriş işleme sistemi ile bir hatayı araştırıyor olabilir, bu yüzden giriş işleme API'sine bağlanan bir kod arayan kodu ararlar. Açıkça, eğer programcı eski giriş işlem koduna rastlarsa, düzenlemeye ve hata ayıklamaya başlayabilirler, çalıştıkları kodun hiçbir zaman yürütülemediğini ve bu nedenle çözmeye çalıştıkları sorunun bir parçası olmadığını fark etmeden önce değerli zaman harcarlar. . Diğer problemler arasında, daha uzun derleme süreleri ve programcıların yanlışlıkla çalışma kurallarını yanlışlıkla yeniden diriltebilme riski vardır. Kaynak kodunda tekne ankrajları ile uğraşmak için önerilen bir çözüm, bunları kod tabanından kaldırmak ve gerekirse ayrı bir yere yerleştirmek, ancak derlenmeyecek veya "çalışma" koduyla karıştırılmayacak şekilde yerleştirilmelidir. (Örneğin, projenin kaynak kontrolünde saklandığını bilerek bunları silme)

3.CORNCOBS

Mısır koçanı yazılım geliştirme işinde yaygın olan insanlar için zordur. Bu tutum, bireysel kişiliğin yönlerinden kaynaklanabilir, ancak sıklıkla, tanıma veya parasal teşvikler için kişisel motivasyonlardan kaynaklanan zorluklar ortaya çıkar.
Sıkı programlar ve bütçe baskısı nedeniyle, yazılım geliştirme stresli hale gelebilir. Mısır koçanı bu sorunları şiddetlendirir ve zaten aşırı stresli bir ortamın olabileceği gereksiz yere ek stres yaratır. "Corncob", zor insanları tanımlamak için bir yazılım konsorsiyumu olan Object Management Group'ta sıklıkla kullanılan bir argo terimidir.

Zor bir kişi (Corncob), bir yazılım geliştirme ekibinin yıkıcı davranışlarıyla veya hatta daha da kötüsü, bir kuruluş boyunca sorunlara neden olur. Bu kişi, ekibin bir üyesi ya da (teknik bir mimar ya da geliştirme yöneticisi gibi) teknik eleman, politik ve kişisel olarak çeşitli yollarla ekibi olumsuz etkileyen dış kıdemli personelin bir üyesi olabilir.
Corncobs'la uğraşırken, siyasetin iktidarın kullanılması olduğunu hatırlamak önemlidir ve Mısır koçanı politikadan çok teknolojiden çok daha fazla odaklanmaktadır. Onlar genellikle siyasetin kişisel ve / veya örgütsel düzeyde manipüle edilmesinde uzmanlardır. Teknoloji odaklı insanlar, Corncob'un taktiklerinin kurban edilmesini kolaylaştırabilirler.

Semptomlar ve Sonuçları
Bir geliştirme ekibi veya proje ilerlemeyi başaramaz, çünkü birisi temel hedeflerine veya temel süreçlerine katılmaz ve sürekli olarak bunları değiştirmeye çalışır.
Bir kişi sürekli olarak endişe duyulan endişe çerçevesinde itirazlarını performans, güvenilirlik, teknoloji pazar payı vb.
Yıkıcı davranış, işletmedeki pek çok kişi tarafından iyi bilinir, fakat yönetim tarafından tolere edilir ve desteklenir (bir şekilde), çünkü bunların neden olduğu hasarın farkında değillerdir ya da bunlarla ilgilenmek istememektedirler.
Siyasi güçler, teknik tartışmaları yolda sürdürmenin zor olduğu bir ortam yaratır.
Siyasi güçler, sistemin kapsamına veya gereksinimlerine sık sık değişmelere neden olur. Proje proaktif olmaktan çok daha reaktif hale geliyor, çünkü herkes Corncob'dan “sonsuz gelişmelere” yanıt veriyor.
Çoğu zaman, yıkıcı kişi, kıdemli yazılım geliştirme müdürü veya proje yöneticisinin doğrudan yetkisi altında olmayan bir yöneticidir.
Şirketin sorunları çözmesini ve devam etmesini sağlayan tanımlanmış bir karar alma süreci yoktur. Bu, bir yöneticinin kendi sorumluluk alanından uygunsuz şekilde müdahale etmesine izin verir.

Bilinen İstisnalar
Corncob AntiPattern, bir şirket veya ürün geliştirme müdürü, Corncob'un eylemleriyle yaşamak istediğinde kabul edilebilir. Bu, fayda ve dezavantajların öznel bir yargısıdır.
Birden fazla kuruluş içeren projelerde, bazen mevcut bir mimariyi uygunsuz değişikliklerden korumak olan atanmış bir Corncob'a sahip olmak yararlıdır. Çok sayıda çelişen teknik görüş açısı olduğunda, referans mimariyi uygulamak için baskın bir kişilik gerekir.

Refactored Çözüm
Corncob AntiPattern'e yönelik çözümler, stratejik, operasyonel ve taktik dahil olmak üzere çeşitli yetki seviyelerinde uygulanır. Her durumda, kilit eylem kolu, yıkıcı davranışlar için yönetim desteği. Yukarıda belirtilen düzeylerde yönetim desteğini ortadan kaldırarak, Corncob desteği kaybeder ve yazılım geliştirme ekibinin en iyi çıkarları hakim olabilir.

Taktik Çözümler
Taktik çözümler, toplantıda olduğu gibi, uçuşta kullanılan eylemlerdir. Aşağıdakileri içerir:
Sorumluluğu aktarın. Kararlaştırılan bir süre içinde tanımladıkları problemleri çözmek için naysayöre tüm sorumluluğu verin. Başka bir deyişle, planlama ve çözümleme konusundaki sorumluluğu, endişeleri dile getiren kişiye devredin.
Sorunu yalıtın. Sıklıkla, Corncob'ın izole bir fikirdir. Grubun menfaatlerine hitap etmek, bir bireyin davranışlarına veya yalıtılmış kaygılarına karşı savunmanın etkili bir yoludur. Eğer mısır koçanı çatışıyorsa, asla kişisel olarak hiçbir şey almayı unutmayın; Bunu yapmak her zaman bir hatadır. Ana itirazları tartışmak ve çerçevelemek için grubu kolaylaştırın, sonra da itirazcıyı geçersiz kılmak için fikir birliğine varmaya çalışın. Bir saman anketi, grup duygusunu açığa çıkarmak için etkili bir tekniktir.
Soruyu sor. Mısır koçanı belirsiz veya "yüklü" kelimeleri veya cümleleri kullandığında, kendi anlamlarını açıklığa kavuşturmasını isteyin. Bir iddiada bulunmak için duruşma kanıtı kullandığında, iddialarını doğrulamasını ve kişisel pozisyonunu belirlemesini isteyin.

Operasyonel Çözümler
Operasyonel eylemler sınırlı bir organizasyon kapsamı dahilinde çevrimdışına alınır. Bunlar şunları içerir:
Düzeltici görüşme. Yönetim, problemlere neden olan kişi ile bireysel olarak buluşur ve davranışının

etkisini açıklar. Düzeltici görüşmenin amacı, farkındalığı arttırmak ve davranışları nasıl değiştireceğine dair anlaşmaya varmaktır.
Dostu deplasman. Bir kurumda zor bir durumdan çıkmak için kendisine yardım etmesi için bir iş bulmacasına bir Corncob tavsiye et.

Stratejik Çözümler
Stratejik eylemler uzun vadelidir ve aşağıdaki gibi daha geniş bir işletme kapsamına sahiptir:
Mısır koçanı destek grubu. Bir organizasyonda birkaç Corncobs varsa, yönetim bunları aynı gruba aktarabilir. Bu şekilde birbirlerinin kişilikleri ve davranışları ile savaşmak zorundadırlar. Çoğunlukla, her biri yönetime yaklaşır ve neden bu kadar zor insanlarla çalışmak zorunda olduğunu sorar, yöneticiye, kişinin kendisi tarafından başkaları tarafından zor olarak nitelendirildiğini açıklamak için açıklık verir. Bu, öz farkındalık ve belki de iyileştirme ile sonuçlanabilir.
Boş bölüm. Kendileri zor olan yöneticiler, tek çalışan oldukları bölümlere aktarılabilirler. Bu yöneticiler genellikle "mesajı alırlar" ve diğer şirketlerde emekli olur veya kariyer peşinde koşarlar.
Kuvvette azalma. Bazen proje ekibinden veya çevreden gelen zor insanları ortadan kaldırmaktan başka bir yol yoktur.

Diğer Bakış Açıları ve Ölçeklerine Uygulanabilirlik
Corncobs'un mimarlık ve kalkınma üzerindeki etkisi genellikle ilerlemeyi yavaşlatmaktır. Corncobs tarafından ortaya atılan çatışmalar, tartışmanın karmaşıklığını azaltma ve bunun yerine belirli güçlere odaklanma eğilimindedir.
Bir anlamda, Corncob davranışları tasarım modellerinin amaçlanan faydalarını tersine çevirir. Yazılım sürecini daha sofistike ve verimli hale getirmek yerine, Corncobs daha az sofistike ve daha az verimli hale getirir.
Sürecin daha dengeli hale getirilmesi yerine, tüm önemli kuvvetler dikkate alınarak, Corncob tartışma ve karar verme sürecini, en önemli olmayabilecek, seçilmiş kuvvetlere yöneltme eğilimindedir.