

## Yazılım Geliştirme Mitolojisi: Efsaneler ve Gerçekler

Dr. Hakan Erdogmus  
Kalemun Research Inc.

YKGS 2010, 4 Aralık, 2010

- 
- ▶ 1. Yazılım projesi yönetimi
  - ▶ 2. Kalite ve verimlilik



# NATO Software Engineering Conference Garmisch, 1968



# Yazılım krizi



# Yazılım krizinin nedenleri ve çözümü

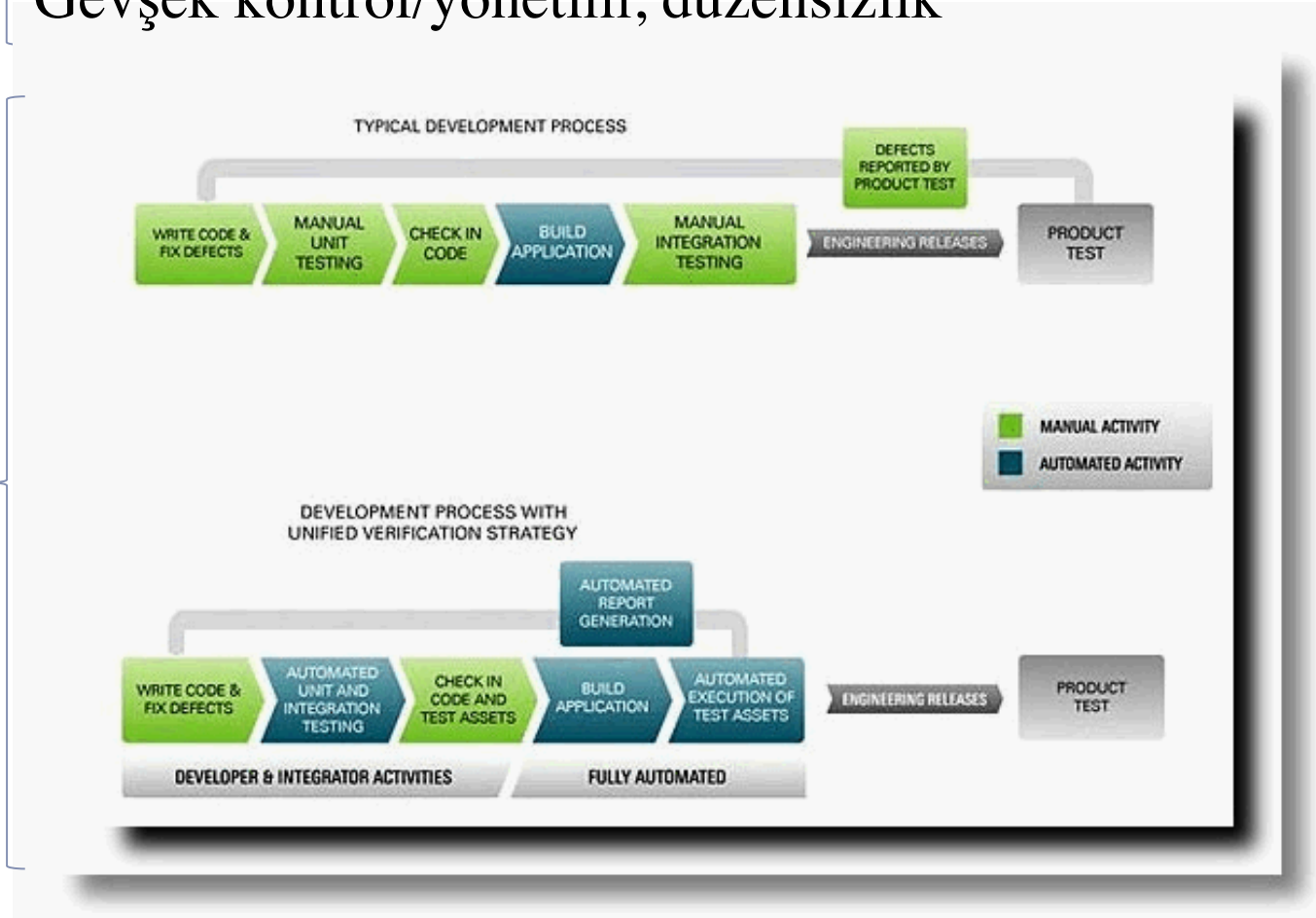
*Sebepler*

~~Teknik becerisizlik~~

Mühendisliğe dayalı olmayan yaklaşım

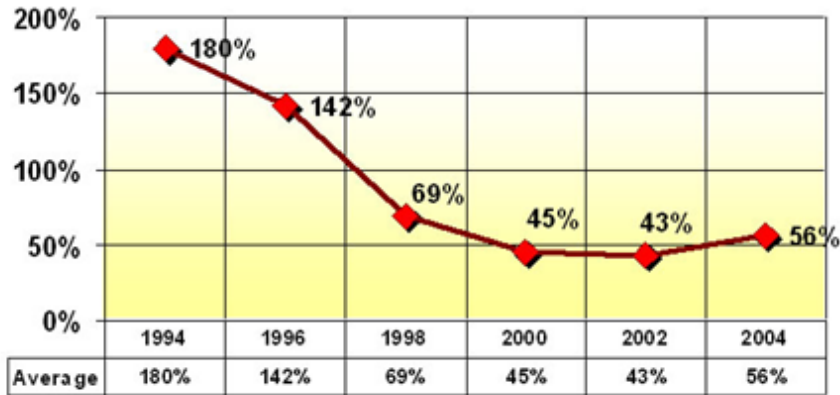
Gevşek kontrol/yönetim, düzensizlik

*Çözüm*



# Standish Grubu CHAOS raporları

1994-2004 Average Percent of Cost Overrun

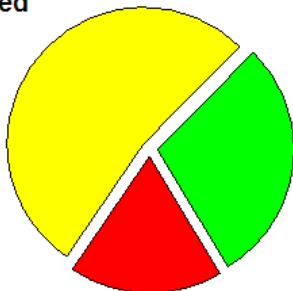


## CHAOS 2004

SURVEY RESULTS

Resolution of Projects

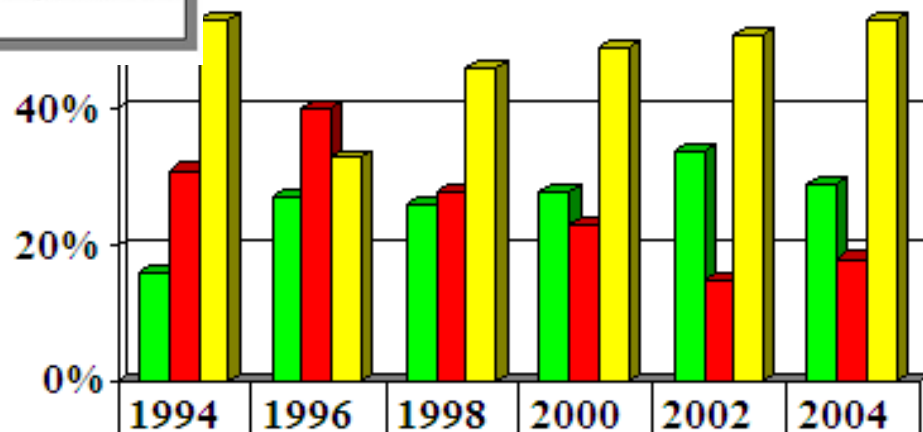
Challenged  
53%



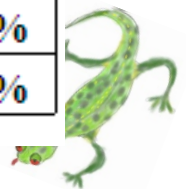
Succeeded  
29%

Failed  
18%

Copyright © 2006 The Standish Group International, Inc..



Succeeded	16%	27%	26%	28%	34%	29%
Failed	31%	40%	28%	23%	15%	18%
Challenged	53%	33%	46%	49%	51%	53%



# CHAOS raporları neye dayanıyor?

---

- ▶ Robert Glass, “The Standish Report: Does it really describe a software crisis?”, CACM, 2006
- ▶ Magne Jørgensen and K. J. Moløkken-Østvold, “How Large Are Software Cost Overruns? Critical Comments on the Standish Group’s CHAOS Reports,” IST 2006
- ▶ <http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS>



# Başarıyı nasıl tanımlamak gerekir?

---



- ▶ Proje başarı oranları bu kriterlere göre düşük olabilir: %50'lerin altında
- ▶ Hedef: *tekrarlanabilirlik ve önkestirebilirlik*



## Şu proje başarılı mı?

---

- ▶ Bütçeyi %100 aşmış
- ▶ %100 gecikmiş
- ▶ Öngörülen işlevlerin sadece %10'u planlandığı biçimde teslim edilmiş ve %20'si önemli bir şekilde değiştirilmiş
- ▶ Ürün organizasyon içinde binlerce çalışan tarafından verimliliği %20 arttırarak her gün kullanılıyor, hem de asgari destekle
  - ▶ Proje bedeli: 4 Milyon Dolar
  - ▶ Yıllık tasarruf: 2 Milyon Dolar
  - ▶ Yatırım geri dönme süresi: 2 yıl



## Efsane #1

---

**Dikkatli, detaylı, ve tam yapılmış bir yazılım geliştirme planı daima yürütülebilir, ve bunun sonunda proje başarılı olur**



# Gerçek #1

---

## Yazılım geliřtirmenin doğal özellikleri:

- karmaşıklık
- belirsizlik (risk)
- yaratıcılık



## Efsane #2

---

**Yazılım geliştirme projesinin başarısı projenin önceden belirlenen bütçeye ve plana uygun olarak bitirilmesiyle ilgilidir**



## Gerçek #2

---

**Yazılım geliştirme projesinin başarısı  
projenin faydası ve yarattığı net değer  
ile ölçülür**



Diğer gerçekler

---

**Düşük giriş bariyeri**

**Yazılımların sosyal ve ekonomik hayatın her alanındaki muazzam etkisi ve yaygınlığı**



## Efsane #3

---

**Yazılım geliştirme sektörü 1960  
yıllarından beri daimi bir kriz altında**



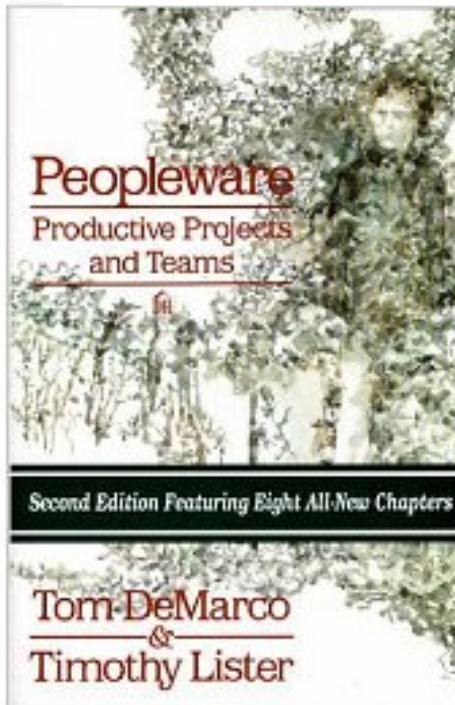
## Efsane #4

---

**Yazılım krizini çözmek için yeni süreçlere, teknolojilere, araçlarla, standartlara, programlama dillerine ve modelleme yöntemlerine gerek var**



# Kontrol?



## viewpoints

Contact Editor: Dennis Taylor ■ dtaylor@computer.org

# Software Engineering: An Idea Whose Time Has Come and Gone?

**Tom DeMarco**

**W**e're now just past the 40th anniversary of the NATO Conference on Software Engineering in Garmisch, Germany, where the discipline of software engineering was first proposed. Because some of my early work became part of that new discipline, this seems like an appropriate moment for reassessment.



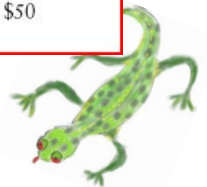
My early metrics book, *Controlling Software Projects: Management, Measurement, and Estimation* (Prentice Hall/Yourdon Press, 1982), played a role in the way many budding software engineers quantified work and planned their projects. In my reflective mood, I'm wondering, was its advice correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort? My answers

### Compelled to Control

The book's most quoted line is its first sentence: "You can't control what you can't measure." This line contains a real truth, but I've become increasingly uncomfortable with my use of it. Implicit in the quote (and indeed in the book's title) is that control is an important aspect, maybe the most important, of any software project. But it isn't. Many projects have proceeded without much control but managed to produce wonderful products such as GoogleEarth or Wikipedia.

To understand control's real role, you need to distinguish between two drastically different kinds of projects:

- Project A will eventually cost about a million dollars and produce value of around \$1.1 million.
- Project B will eventually cost about a million dollars and produce value of more than \$50 million.



## İki proje (DeMarco, 2009)

---

### ▶ Proje A:

- ▶ Yatırım: 1 milyon dolar
- ▶ Fayda: 1.1 milyon dolar

### ▶ Proje B:

- ▶ Yatırım: 1 milyon dolar
- ▶ Fayda: 50 milyon dolar

Hangisini kontrol etmek daha önemli?



# Uç bir örnek: TikiWiki

from the editor

*IEEE Software,  
Nov/Dec 2009*

Editor in Chief: Hakan Erdogmus ■ Kalemun Research ■ hakan.erdogmus@computer.org

## A Process That Is Not

**Hakan Erdogmus**

**T**ikiWiki, or Tiki in short, is a notable piece of software. It's open source, big, successful, and widely used. Nothing too special about that. But Tiki also embraces Erik Raymond's "bazaar" model ([www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar](http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar)) in the extreme. It's not driven by a handful of core developers, or supported by an ecosystem of third-



party contributions that plug in to the core. It doesn't have any systematic quality controls: no design reviews, no testing, no real gatekeepers, no architecture workshops, no imposed architecture, nothing. It doesn't have an ad hoc steering committee making quasi-binding decisions about what's important and what's not, or who's allowed to touch what, or veering the project strongly in this direction or that. Granted, at any given time, it does have its vision-

a wiki engine. It's used in Web sites, both commercial and nonprofit, in various ways: as a plain wiki, a wiki on steroids, a content management system, a groupware application, a Web application, a Web portal, an issue-tracking system, a form generator, a knowledge base, and combinations thereof. A close colleague, Alain Desilés, first brought Tiki to my attention. Alain is a contributor himself, and was amazed at how the Tiki approach works on the scale that it does. To reassure those who are unfamiliar with Tiki that it's not marginal software with marginal success, here are some facts.

### **Tiki Facts**

Tiki was first released in 2002 and has been under active development for seven years. It now has more than a million lines of PHP code and more than a thousand features and configuration options. With over 200 active source-code contributors, Tiki has one of the largest open source teams in the world.



## Efsane #5

---

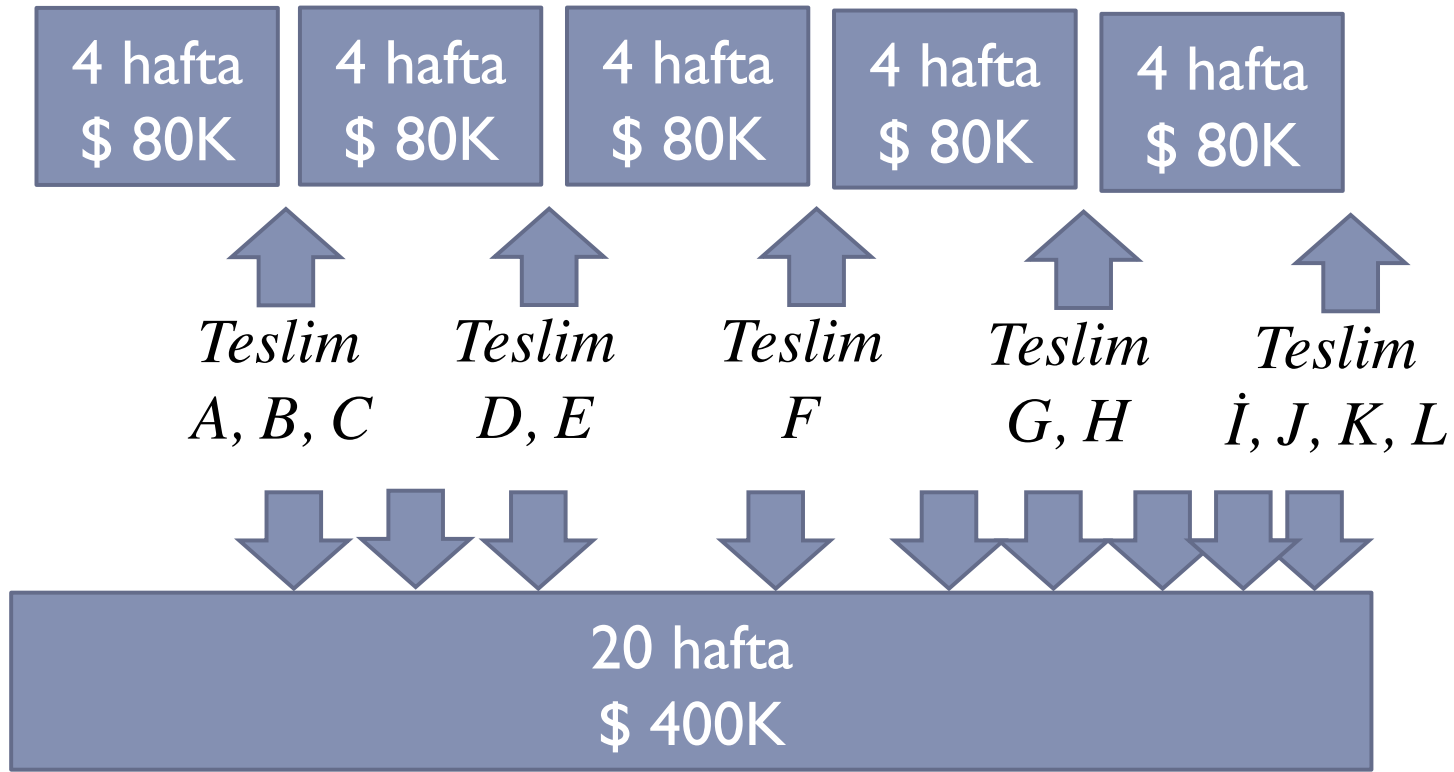
**Riskli, büyük yazılım projelerinin başarısı için sadece sıkı kontrollü resmi süreçler etkilidir ve gereklidir**



# Esnek yönetim yaklaşımı

- ▶ **Sabit, kısıtlı:** para, zaman, insan kaynakları
- ▶ **Değişken, kısıtsız:** işlev, içerik

## Artımlı geliştirme



## Gerçek #3

---

**Yazılım geliştirme projelerinin riski  
yok edilemez ancak  
artımlı planlama ve teslim  
ve  
esnek bir yönetim yaklaşımı  
ile indirgenebilir**



---

# VERİMLİLİK VE KALİTE



# Kalitenin hatırı için kalite

---

▶ John Favaro:

“When the pursuit of quality destroys value”

*IEEE Software, May 1999*

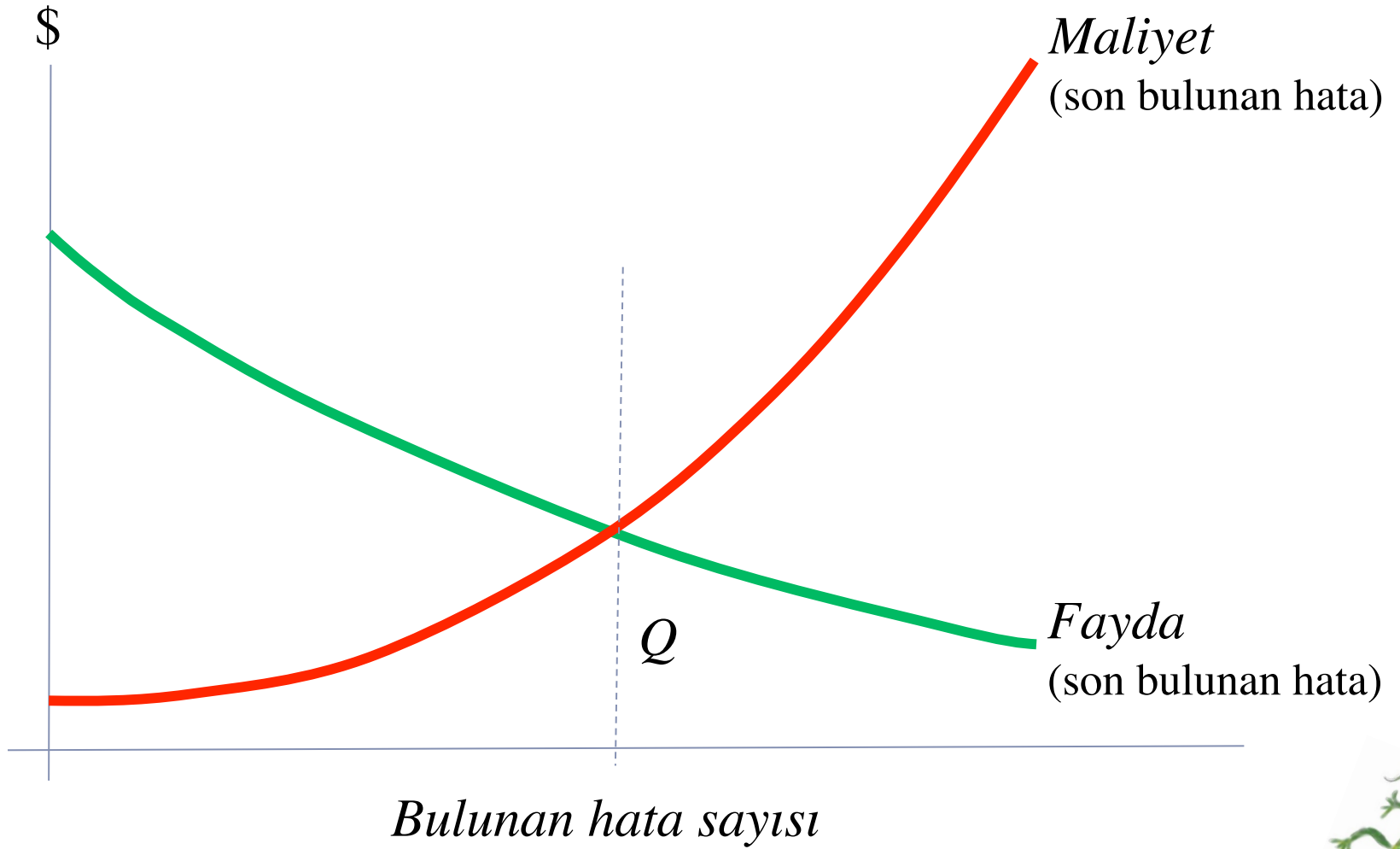
▶ Robert Glass:

“Residual errors will always persist. The goal should be to minimize severe errors”

*Facts & Fallacies of Software Eng., 2003*



# Kalitenin eksilen faydası



## Gerçek 4#

---

**Her ürünün optimal bir kalite seviyesi vardır**



## Efsane #6

---

**Ürün kalitesini arttırmak daima  
projenin net değerini artırır**

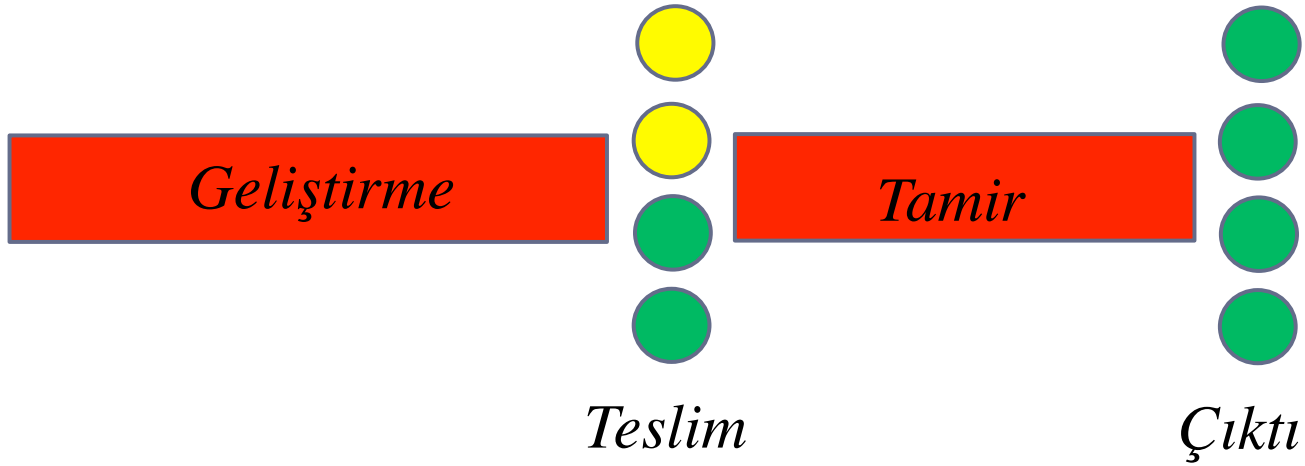


# Verimlilik ve ürün kalitesi arasındaki ilişki

---



# Kısa vadeli (brüt) verimlilik (Süreç 1)



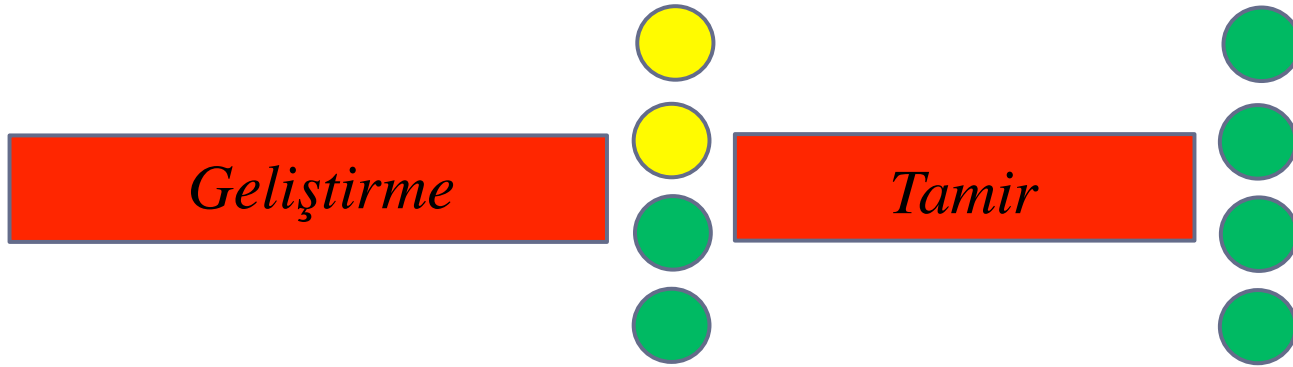
$$\text{Verimlilik} = \frac{\text{Çıktı}}{\text{Geliştirme}}$$

The equation shows the efficiency calculation. The numerator is represented by four circles (two yellow and two green) and the denominator is a red box labeled *Geliştirme*.



# Uzun vadeli (net) verimlilik (Süreç 1)

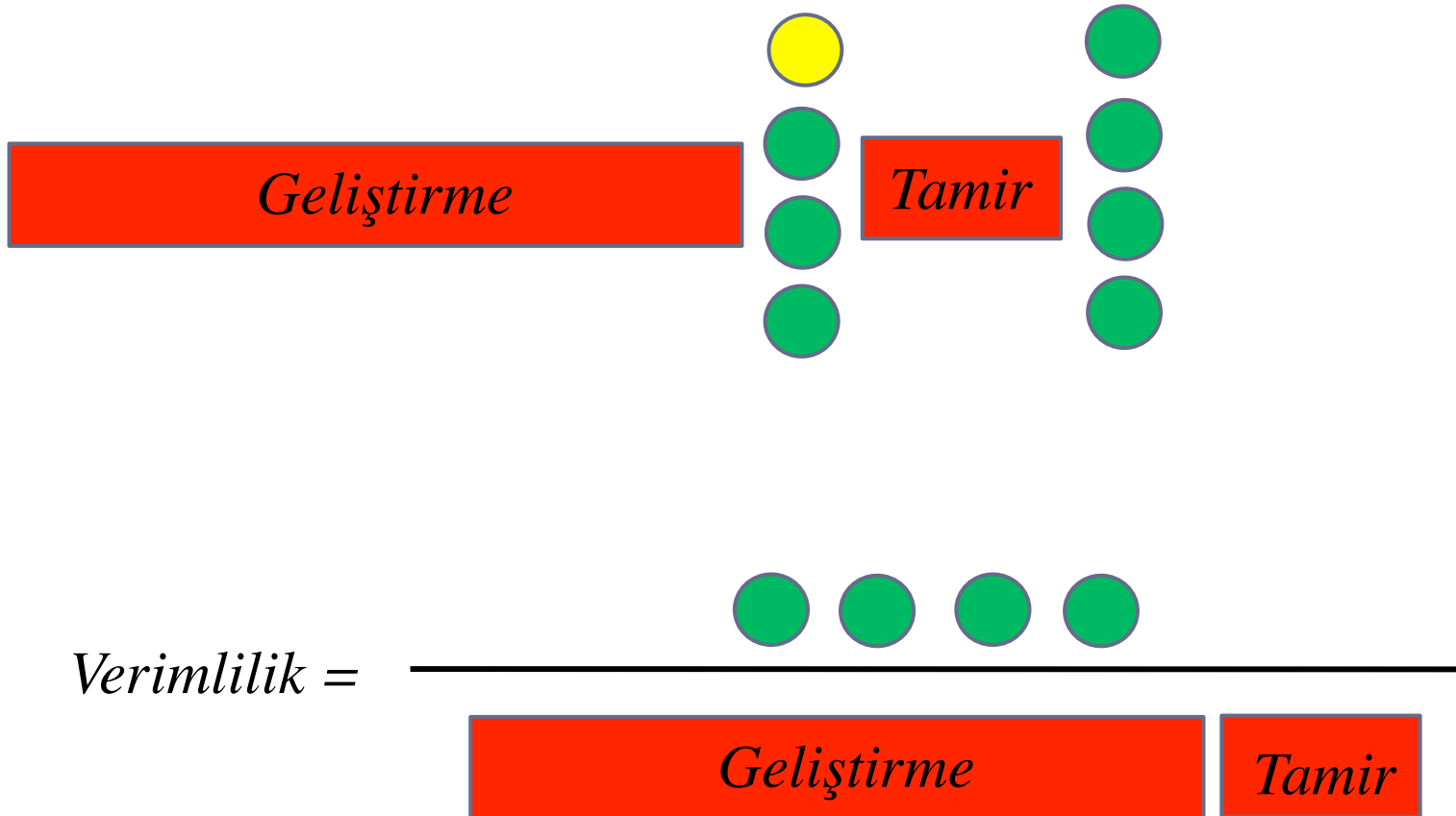
---

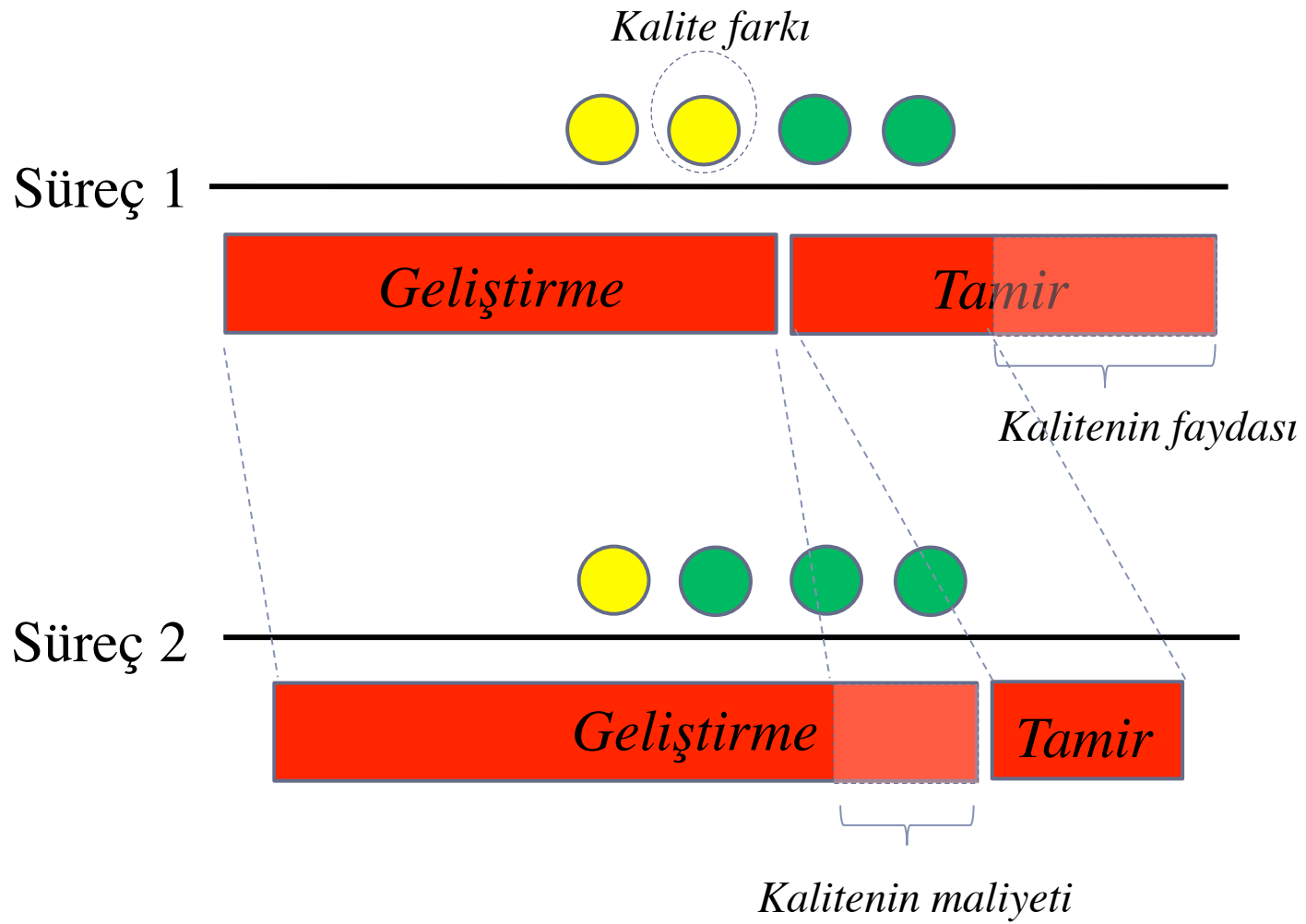


$$\text{Verimlilik} = \frac{\text{Geliştirme} + \text{Tamir}}{\text{Geliştirme} + \text{Tamir}}$$



# Uzun vadeli (net) verimlilik (Süreç 2)





## Efsane #7

---

**Verimlilik ve kalite daima ters orantılıdır**



## Gercek #5

---

**Kaliteyi arttıran bir yöntem  
verimliliği de artırabilir**



# Verimlilikte insan faktörün rolü

---



= 10 x



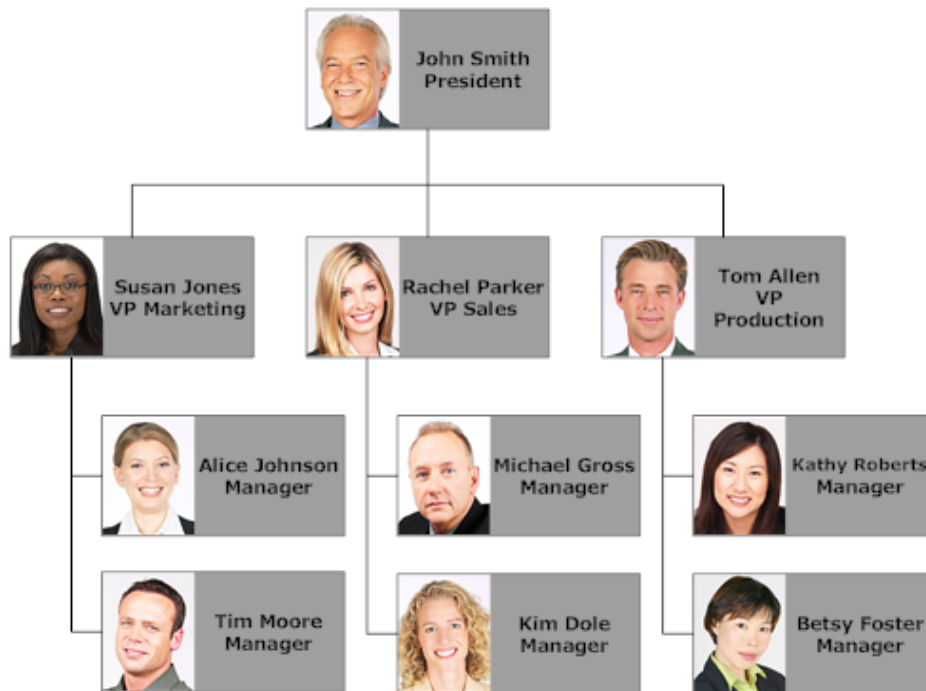
= 3 x



# Conway'in kuralı

---

Arbor Business Company, Inc.



*myapp.arbor.com*

*VpMarketing.java*

```
public Manager alice();  
public Manager tim();
```

*VpSales.java*

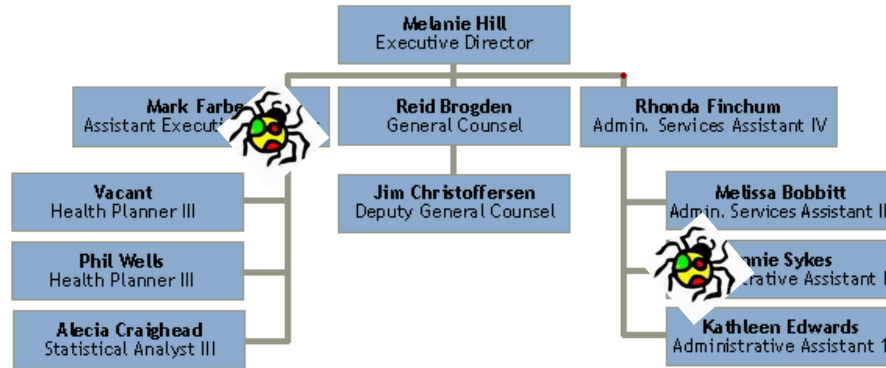
```
public Manager michael();  
public Manager kim();
```

*VpProduction.java*

```
public Manager kathy();  
public Manager betsy();
```



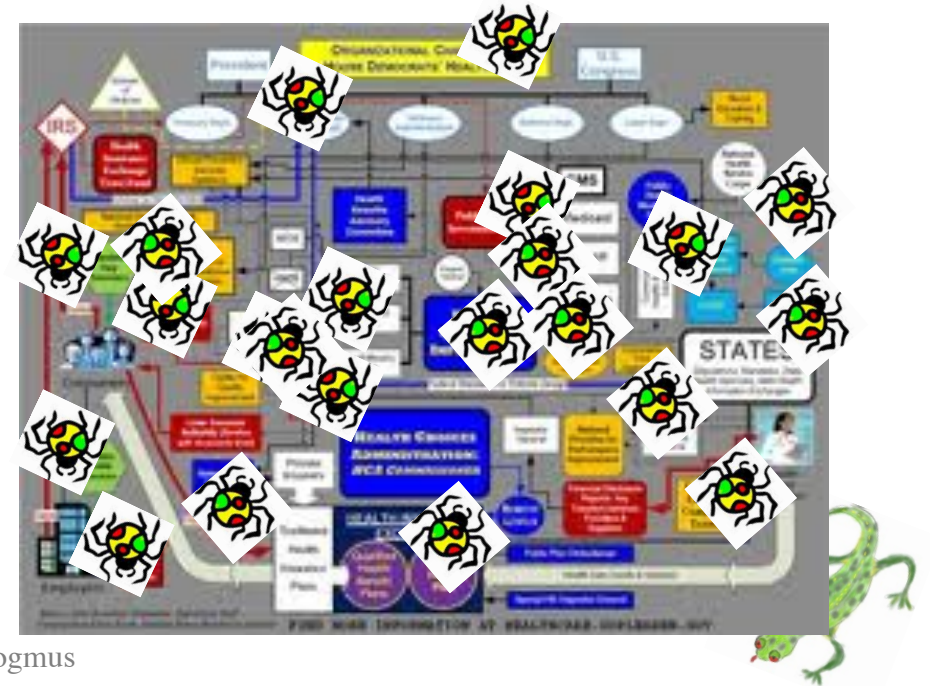
# Yazılım hataları, verimlilik, ve organizasyon yapısı



- ▶ McConnell: Kurumsal faktörlerin verimliliğe etkisi kişisel farklılıklar kadar kuvvetli olabilir

- ▶ Nagappan & Ball: Organizasyonun ve takımın sosyal yapısı yazılım hatalarını en iyi önkestirebilen etken

Making Software, 2010



## Gerçek #6

---

**Yüksek verim ve kalite için uygun süreç ve en iyi gelişimcileri seçmek yeterli olmayabilir**



# Ne yöntemler/kavramlar gerçekten *genelde* etkili?

---

- ▶ Artımlı geliştirme/teslim\*
- ▶ Kod/tasarım gözden geçirme yöntemleri \*\*\*
- ▶ Sınamaya dayalı geliştirme (*test-driven development*) \*\*
- ▶ Çiftli programlama (*pair programming*) \*\*
- ▶ Modülerlik \* \*
- ▶ Tasarım şablonları (*design patterns*)\*\*



# Efsaneler

---

1. Planlama = Başarı
2. Başarı = Önkestirilebilirlik + Tekrarlanabilirlik
3. Yazılım krizi var
4. Yazılım krizi çözülür
5. Risk + Büyüklük  $\Rightarrow$  Başarı = Süreç + Kontrol
6. Kalite = Değer
7. Kalite  $\propto$  1 / Verimlilik

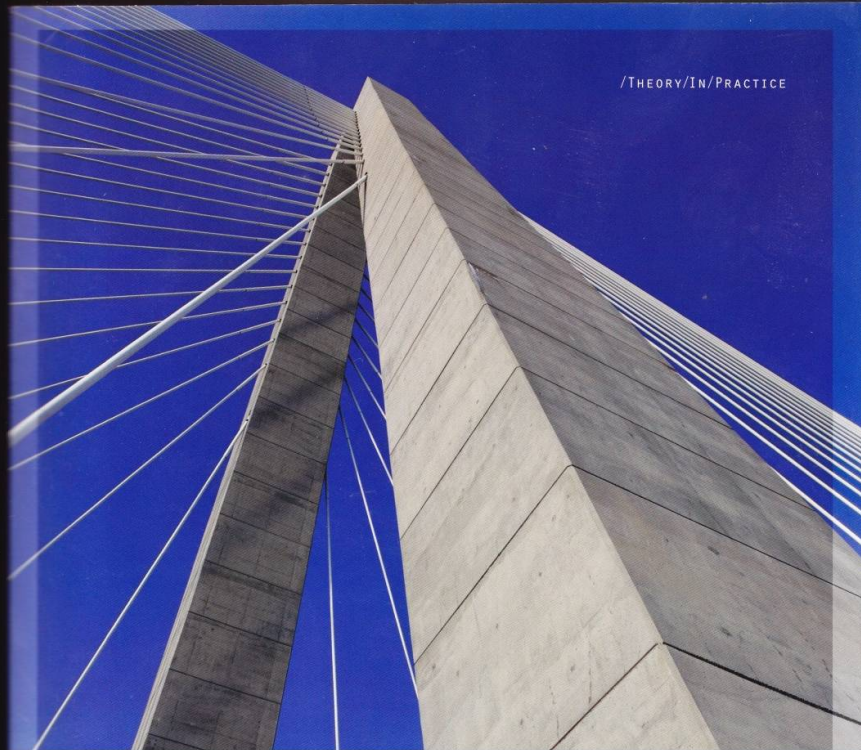


# Gerçekler

---

1. Daimi belirsizlik, risk
2. Başarı = Değer
3. Belirsizlik  $\Rightarrow$  Esnek yönetim
4. Optimal kalite
5. Verimlilik  $\propto$  Kalite? Olabilir
6. Verimlilik ve kalite  $>$  Süreç + Kalifiye eleman





/THEORY/IN/PRACTICE

# Making Software

What Really Works, and Why We Believe It

O'REILLY®

Edited by  
Andy Oram & Greg Wilson

# Facts and Fallacies of Software Engineering



**Robert L. Glass**  
Foreword by Alan M. Davis

