



# Announcing Certification Kamp!

## Buy One! Get One Free!

Now you can get **certification** for ACM, IBM, Oracle, IEEE, Agile Alliance, Harley Davidson, Google, Apple, Orange, SOA, SOB, SOC, Yahoo, FaceBook ... for just **\$99.99**. Learn **all the buzzwords** you could possibly stomach.

**Don't worry about competence** – there's absolutely **no code to write – ever!**

**Just get certified! Bring a friend** and we'll certify the **pair of you!**

**It's all you will ever need for BigCompany.com**

**Just Click CerifyMeNow.tv**

**Click in the next 20 minutes** and we send you a **free** deck of official **Story Cards PLUS** an **XP Hat with Flashing Build Light!** Remember we **tattoo your certificate**, so you are **qualified for life** – *additional charge for foreheads and private parts*. And we **post your photo** on our FaceBook site so that you can **tweet our global followers!** Even your **grandmother** will know **you're certified!**

For groups of **5 or more**, we'll provide you with personalized **“Honest, <your name here>'s Certified” stickers** for your task board.

**No one will ever question your competence again!**

**Show Them You've Got What It Takes in IT Today!**

# GOTO Conference – Notice and Disclaimer

**Warning** – This talk is intended for a **mature** audience with a good sense of **humor**.

**Reproduction in any form:** audio, **video**, pencil, blog, phone, tweet of this **talk** is strictly **forbidden** and is a **violation** of the Digital Millennium **Copyright Act**. **Offenders** will be **aggressively refactored** !

All **people**, **companies**, **situations** used in this talk are **fictitious**, any **likeness** is **purely coincidental**

Special thanks to our sponsor **Certification Kamp**



**WARNING!**  
**THIS TALK IS INTENDED FOR A MATURE DEVELOPER AUDIENCE, IT MAY OFFEND SOME OBJECTHOLICS, RELATIONAL ROW FARMERS and AGILISTAs**

**Some statements are forward looking hence they should be verified first hand by competent developers**

**THE OPINIONS EXPRESSED ARE SOLELY THOSE OF THE PRESENTER AND NOT ANY BUSINESSES OR INSTITUTIONS WITH WHOM HE MAY BE ACTUALLY OR ACCIDENTIALLY ASSOCIATED**

# Why Modern Application Development Sucks!

## Death by Objects, Middleware, Relational Databases, Agile...

Dave Thomas



By Clark & Vizdos

© 2007 [implementingscrum.com](http://implementingscrum.com)

# The Promise and Today's Reality of

- Challenges and Professionalism
- Object Technology and Middleware
- Relational Technology
- Agile
- Life After Modern AD?

# AD Backlog and Challenges

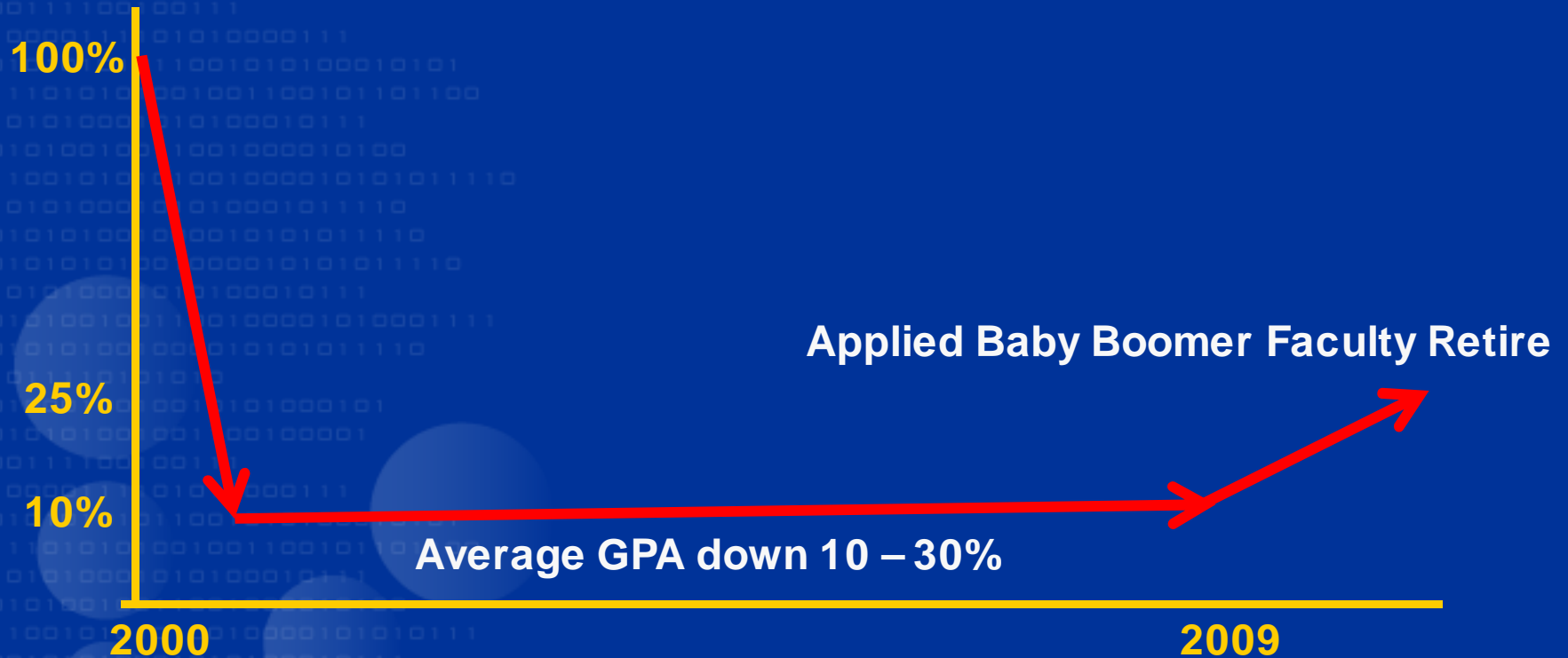
## Application Development Backlog

1. Legacy Evolution (75 – 80%)
2. Government Compliance (5 – 10%)
3. Applications Enabling New Business (10 – 15%)

## Challenges

1. Complexity
2. Skills

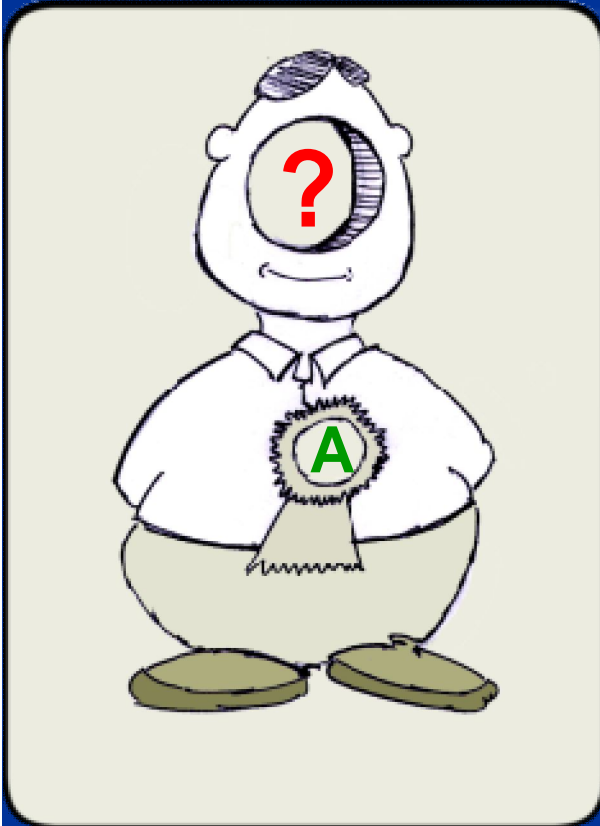
# Uni CS Enrolments 2000 – 2010



Best Students in Business, Bio & Environment Science, Arts  
Virtually no females in CS, Engineering  
Increase focus on certification versus competence  
Talented immigrants return to home countries



# Professionalism – Certification, Accreditation, Self Assessment and Craftsmanship





# The Steps To Enlightenment

IEEE

Journeyman

Accredited

Master

ACM

Apprentice

Meta

Certified



# The Return to Craftsmanship?

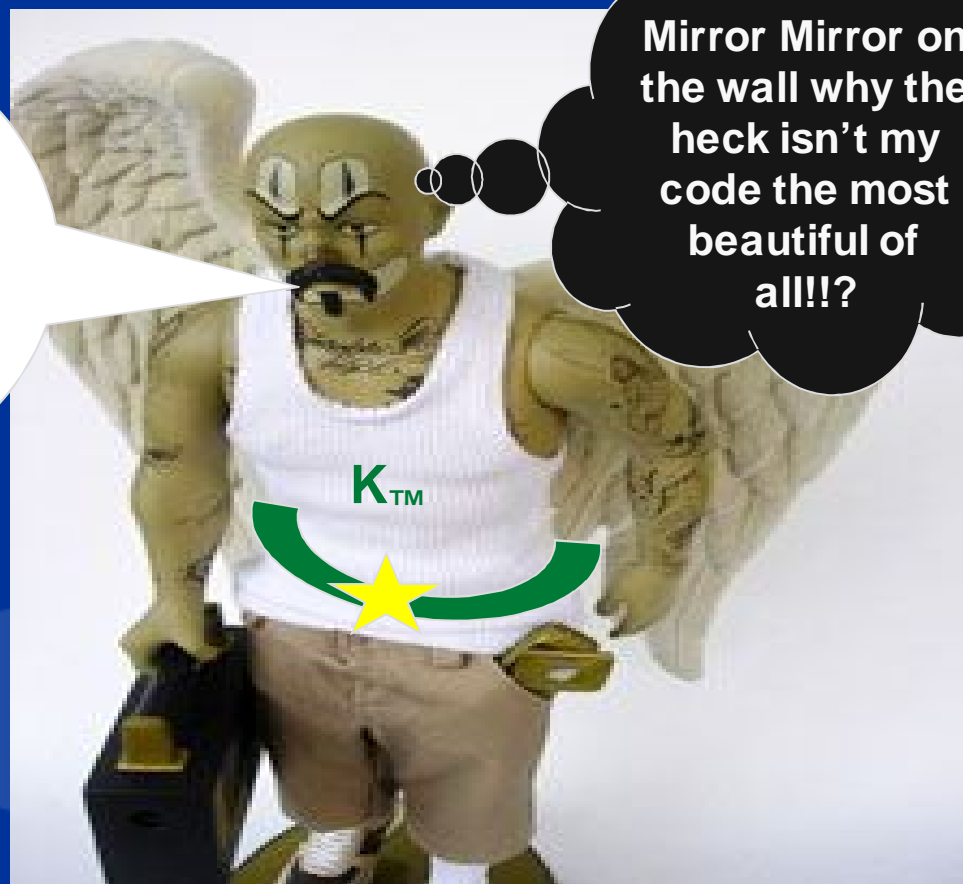


"So you see Bob, an improperly tied knot can ruin a perfectly good hangin'."

# At Last The Craftsman has Arrived!

- **STOP Debt Build Up!**
- **Always** leave the Code **smelling better** than you found **lck!**
- **Don't leave it for your \$%#@# to refactor later!**

Yes! I hate this **illiterate programming**.  
Terrible code smell eh?  
It takes a fully mentored  
**Kraftsman!**  
Oh! Sorry time for my **kata**  
before the code dojo



Mirror Mirror on  
the wall why the  
heck isn't my  
code the most  
beautiful of  
all!?!?

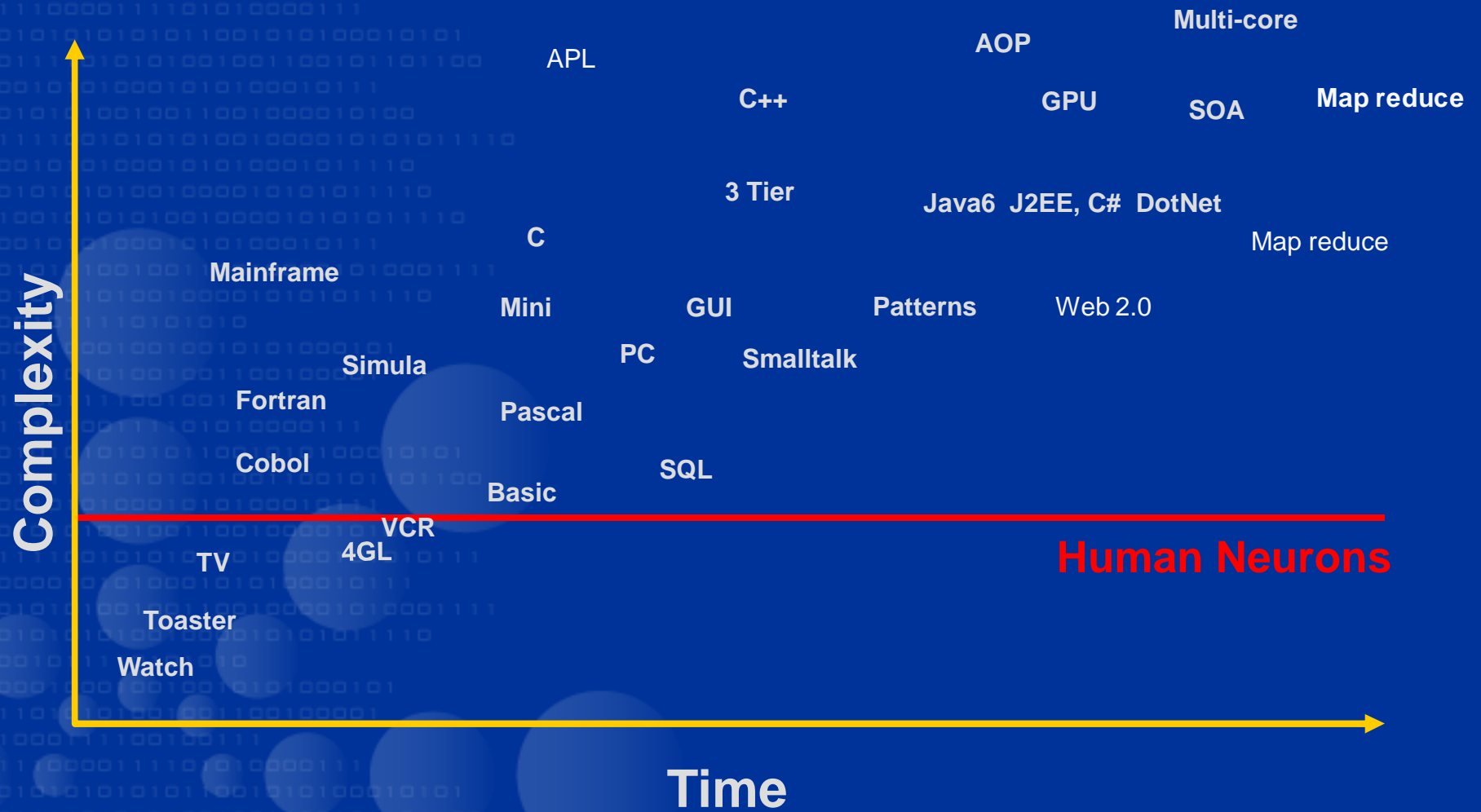


# Jurassic Middleware – Life in The Tar Bits

1. Prehistoric Software As A Service – Mainframe Dinosaurs
2. The Age of Complexity – Client–Server
3. The Age of Naïve Simplicity – PC, Spreadsheets and Enterprise Applications
4. The Age of Absurd Complexity – Objects, Middleware...
5. The Age of Naïve Simplicity – Agile, Rails...



# Complexity of Technology





# The Daunting Difficulty of Application Development

- ↑ API Surface Area = API x Frameworks
- ↑ Language Surface Area = Grammar Productions x Languages
- ↑ Ways of Doing The Same Thing = Platforms x (2 to 4 )
- ↑ API Instability = (Middleware + Upperware + Lowerware) x 3 versions
- ↑ Accidental Complexity
- ↑ Developer IDE Features = Editor + Browser + Build & Test + Versioning + Process + Models x (1–3)
- ↑ Klocs Per App Delivered
- ↑ % of Budget for Maintenance vs. New Development
- ↓ Readability of Code
- ↓ Locality of Application Code
- ↑ Developer Certification versus Competence
- ↑ Global Shortage of IT Skills
- ↑ Vendors Say Life is Getting Better

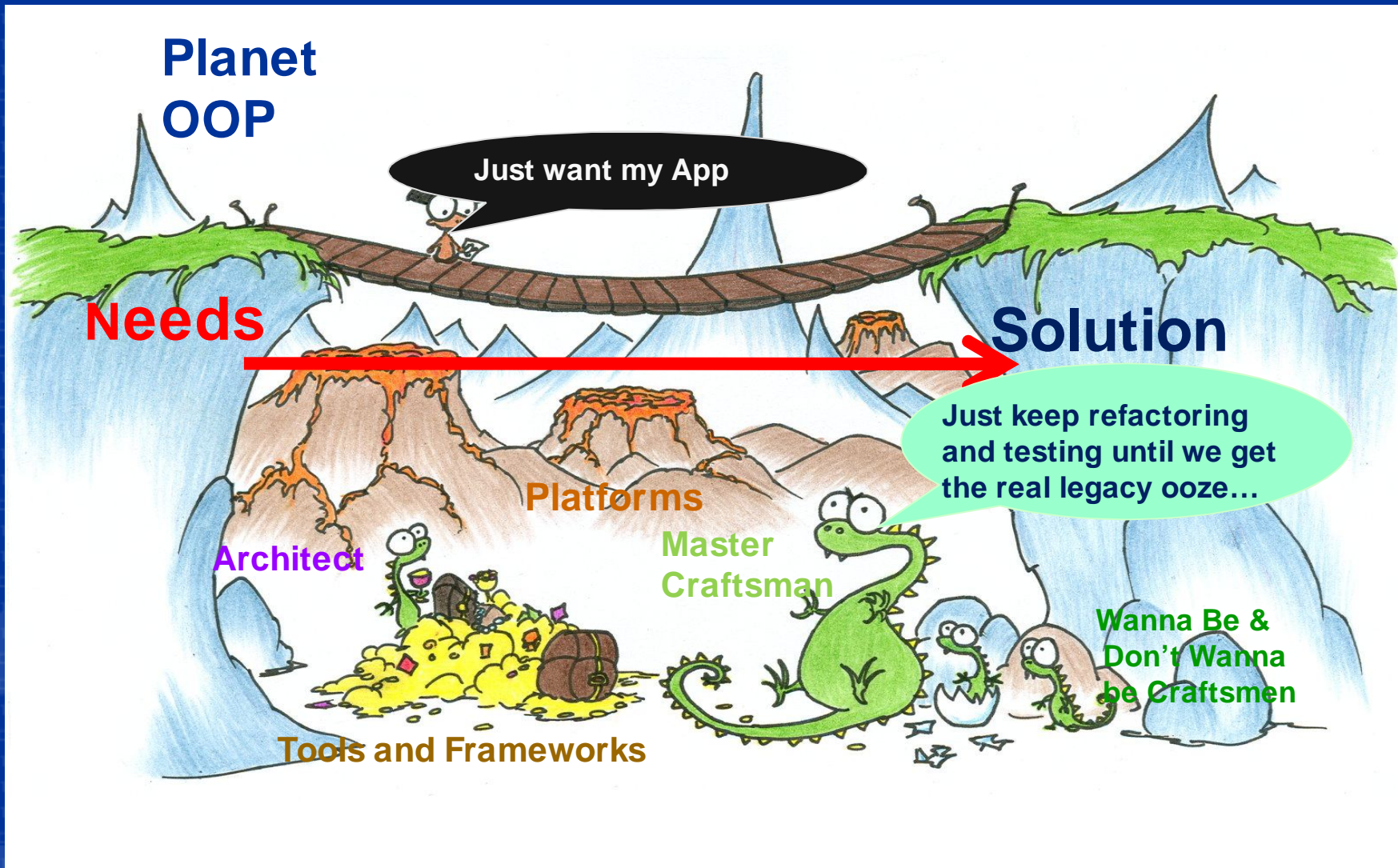
# The Escalating Costs of Ownership

- ↑ Software Tiers
- ↑ Hardware Tiers \* virtualization + clouds ↓
- ↑ Software Stack
- ↑ Software Tools – IDEs, Build, SCM, SQA, Modeling, Performance
- ↑ Installation and Upgrades
- ↑ Vendor Interoperability
- ↑ Enterprise Open Source Version Management
- ↑ Recruiting and Retaining Top Talent
- ↑ Development Maintenance as a % of IT Budget
- ↑ Platform, Framework, Tool churn rate
- ↑ License Complexity and Costs
- ↑ Vendor Lock
- ↑ Vendors say things are getting better

Outsourcing &  
Software  
As A  
Service  
(SaaS)



# Objects are Good, But Are They Necessary or Sufficient?!



# The Darker Side of Objects – Part I

1. Object **design isn't Easy or Natural**  
(business objects, queries, rules, transformations, function composition, algorithms, relationships)
2. **Poor** understanding of **roles, responsibilities, collaborators, contra-covariance**
3. **Interfaces** are still rarely used hence there is a lack of modularity in most **legacy OO** applications
4. Frameworks **infect your code** with their **dependencies**
5. **80%** of applications are **still CRUD, No Domain Model?**



# The Darker Side of Objects – Part II

6. Reuse is a promise not a reality – frameworks not components
7. Libraries APIs poor and code is unstable
8. Object Languages are increasingly complex with attributes, generics, concurrency models, functional enhancements...
9. Objects legacy refactoring is intractable using current tools
10. Serialization is fragile and Slow!

Just add a little glue code and some wrappers to obfuscate those patterns





# “Serialization is Fragile” Josh Bloch JAOO 2009

## Serialization is Fraught with Peril

- Implementation details leak into public API
  - Serialized form derived from implementation - Instances created without invoking constructor
  - Constructors may establish invariants, and instance methods maintain them, yet they can be violated
- Doesn't combine well with final fields - You're forced to make them nonfinal or use reflection
- The result: increased maintenance cost, likelihood of bugs, security problems

**There is a better way! The Serialization Proxy Pattern or DO IT YOURSELF!**

Design a struct-like proxy class that concisely represents logical state of class to be serialized

- Declare the proxy as a static nested class
- Provide one constructor for the proxy, which takes an instance of the enclosing class – No need for consistency checks or defensive copies

**Truth in Advertising *The Serialization Proxy Pattern is not a Panacea!***

- Incompatible with extendable classes and with some classes whose object graphs contain circularities
- Adds 15% to cost of serialization/deserialization

**But when it's applicable, it's the easiest way to robustly serialize complex objects <SIGH!!!>**

# The Darker Side of Objects – Part III

11. A **bulky and computationally expensive runtime abstraction.**
12. Objects at runtime suffer an **acute impedance mismatch** with memory and storage
13. Objects are **slow for modern machines** to execute them efficiently (branch prediction, cache misses, multi-core)
14. OOVMS are **space challenged to share code and data** (**cost of hosting** on OOVMS versus alternatives)
15. Objects are **aggressively sequential and state full** in an increasingly **parallel and immutable** world
16. Object == **identity** is an **expensive** luxury in a **parallel and distributed** world



# Why RDBs Suck?

## SQL

SQL isn't a real language (not Turing Complete)

Real Server Side SQL not standard

Stored Procedures complex and difficult to test

## Relational Model

Normalization unnatural and expensive (e.g. Order and Line Items)

Impedance between Objects and Records (ORM Considered harmful)

Very Limited support (via blobs) complex structured data (objects, xml..)

Lack of Versioning for Schema and Data

## Lack of Agility

Schema Refactoring Expensive; Data Refactoring Expensive

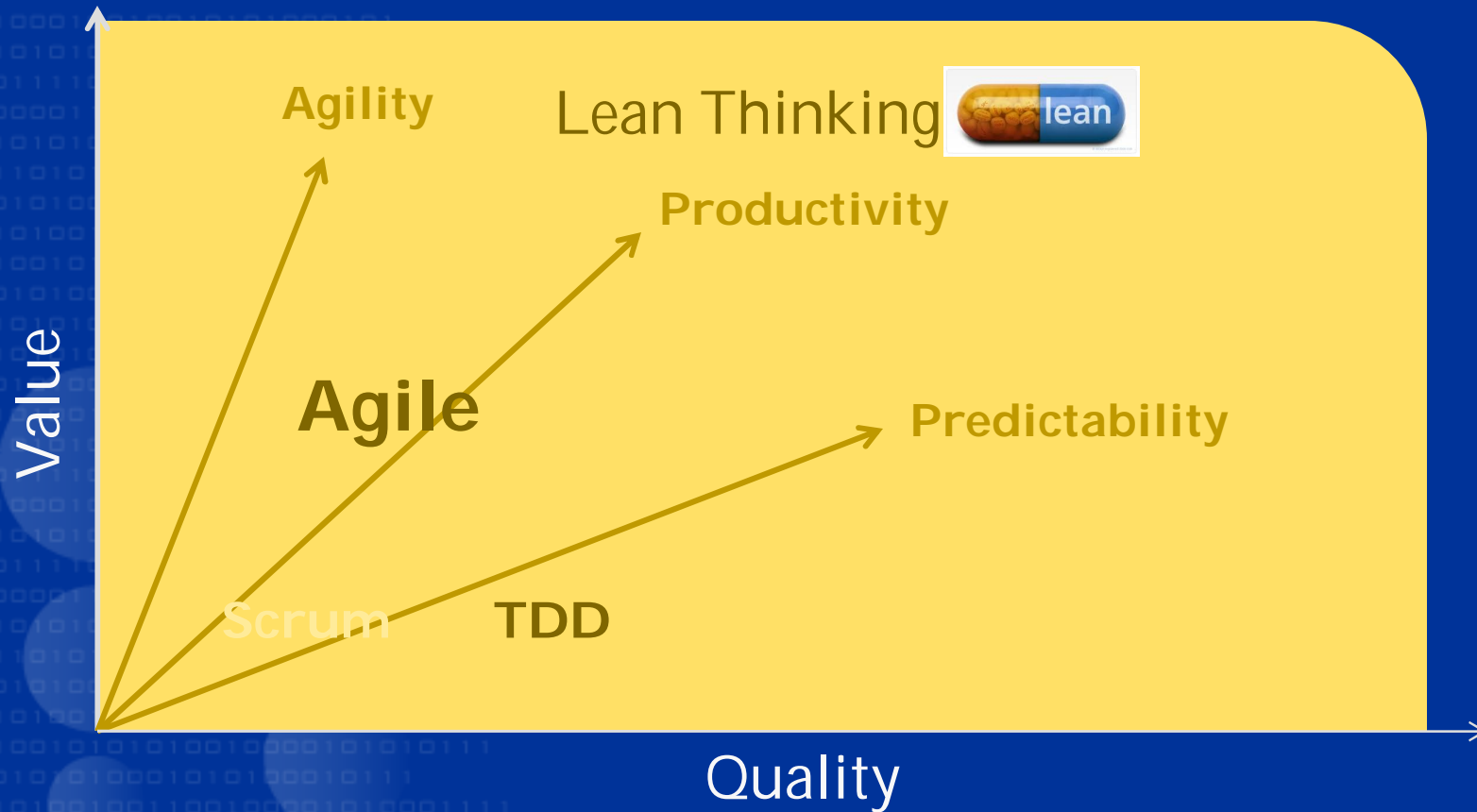
## Limited Horizontal Scale Out

ATOMIC 2 phase transactions expensive

Joins expensive; Row storage expensive



# Agile in The Wild 2011



**Better but not getting what we hoped for  
Agile being deified, diluted, and just practices ignored  
Agile often fails to embrace Lean hence stalls in large co**



# Extreme Messages → Misinterpretations






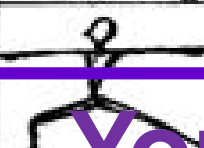

























# Practices! before Method and Tools



## Boarding All Passengers for Flight 42

- process police
- tool fanatics
- bloated framework builders
- enterprise architects
- Naive Agilista

# Practices First! Method is a set of practices

Seated	Kneeling	Standing	Standing	Belly Down	Supine	Inverted
 Bound Angle Baddha Konasana	 Child Garbhasana	 Men. Goddess Devi Tadasana	 Warrior II Virabhadrasana II	 Boat Navasana	 Supine Child Supta Garbhasana	 Supported Shoulderstand
 Staff Dandasana	 Table	 Mountain Tadasana	 Lateral Angle Parsvakonasana	 Sphinx	 Supine Spinal Twist Supta Matrsyendrasana	 1/4 Shoulderstand Ardha Sarvangasana
 Head to Knee Janu Shirshasana	 Pigeon Kapotasana	 Standing Squat/Chair Utkatasana	 Triangle Trikonasana	 Cobra Bhujangasana	 Bridge Setu Bandhasana	 Fish Matsyasana
 Posterior Stretch Paschimottasana	 Downward Dog Adho Mukha Shvanasana	 Warrior I Virabhadrasana	 Half Moon Ardha Chandrasana II	 Frog Mandukasana		
 Seated Spinal Twist Matrsyendrasana		 Tree Ardha Chandrasana I	 Tree Vrikshasana			

Your  
Method

Left  
Method

Right  
Method

# Change is easy, as long as someone else is doing it



A Natural-Born 'Leader' Emerges...apparently self organization needed his leadership



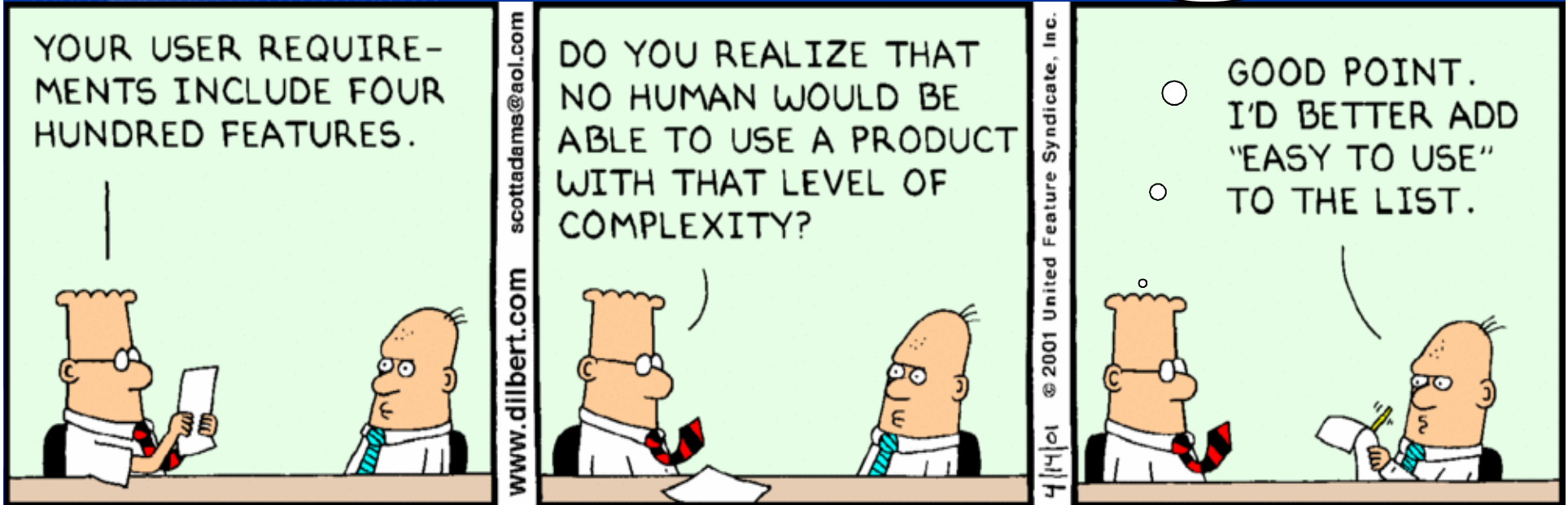
"Who him? ... Oh, he's the Product Owner you've been hired to replace...he failed to feed the developers on demand

~~March Sprint 1 2 3 4, Sprint! .....~~

# Take a look at the state of your backlogs!

- Lumpy Stories – sizes, estimates, lack tangibility, business value
- More stories keep getting added to the backlog
- Blocked waiting on ...
- You mean the customers won't be sitting with each team!

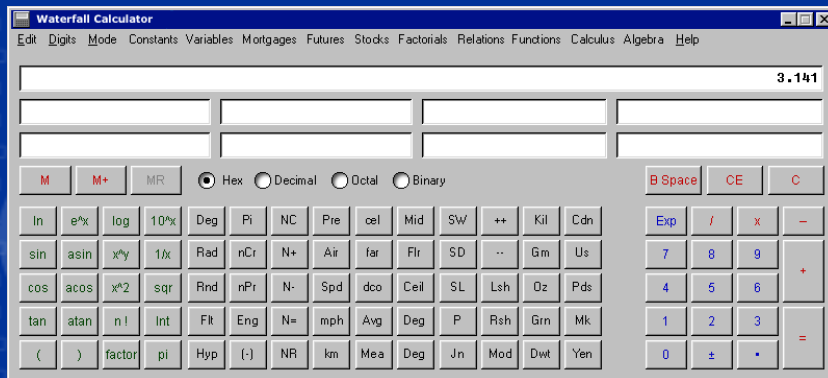
Wow! I want to be a portfolio Manager





# Envisioning: Requirements Thru Design

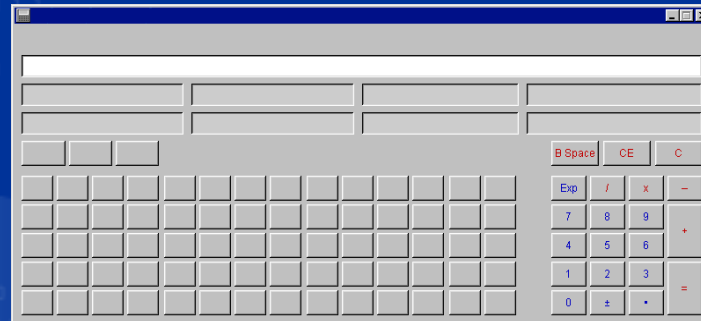
Allows us to understand enough of the vision of 'tomorrow'...



The Waterfall Pitfall

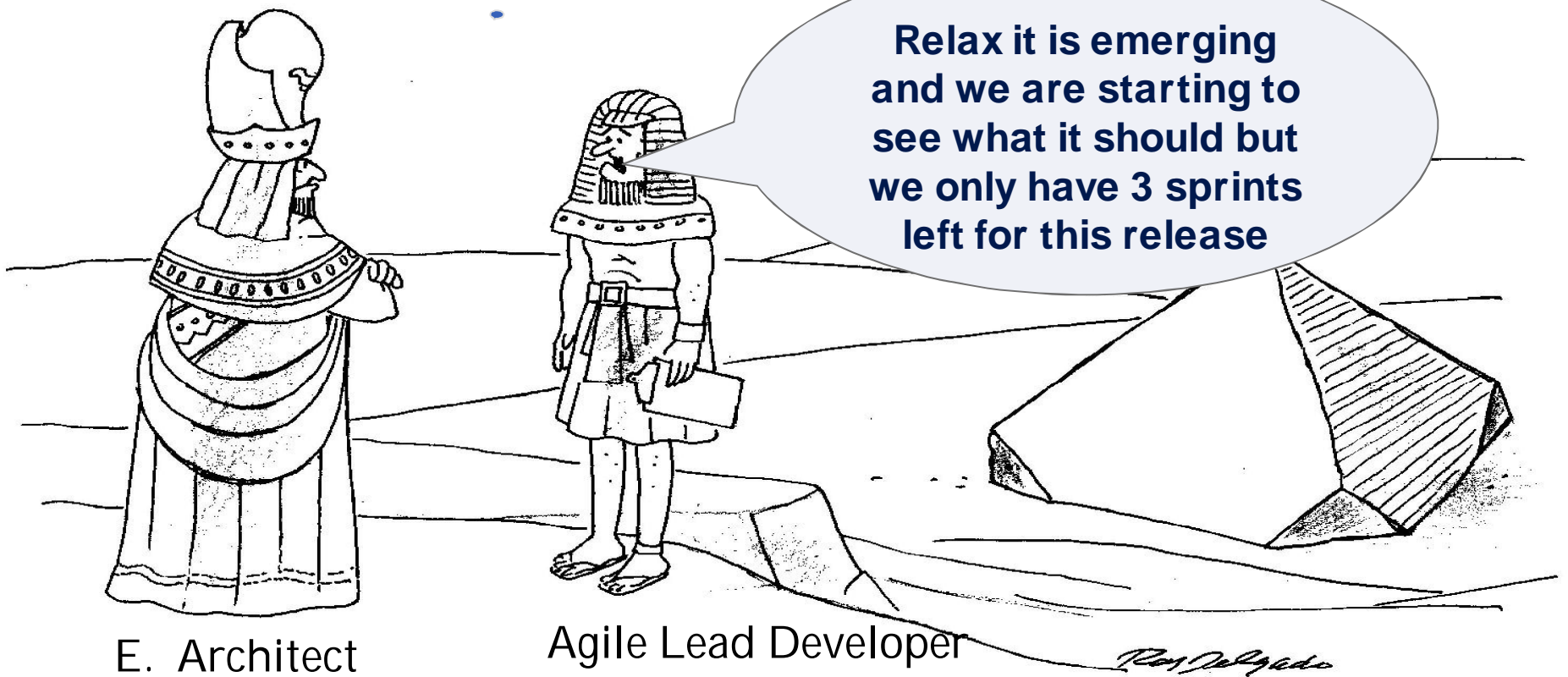


The Agile Pitfall



Agile with Envisioning

# Architecture and Design Makes a Difference





# Sprint0 – It may take more than 1 iteration

Architecture just emerges?

We should have done some Envisioning



Another Sprint Zero off to a great start!

Somewhere around Sprint 23, Rod realizes that his architecture needs a bit more work...

!

# Collective Ownership



Collective Ownership

## 4 eyes on the code

- shares knowledge
- reduces risk
- reduces stress
- prevents defects

Achieved by **pairing** OR  
**reviews, inspections...**

# TDD – Done means Unit and Acceptance Tested!

## Test Economics

- AT value \$\$\$ >>> UT \$
- Test Automation Triage \$\$
- Parametric/Random Testing
- Test Soon vs Test First \$\$

“Testing shows the **presence** not **absence** of bugs”\*



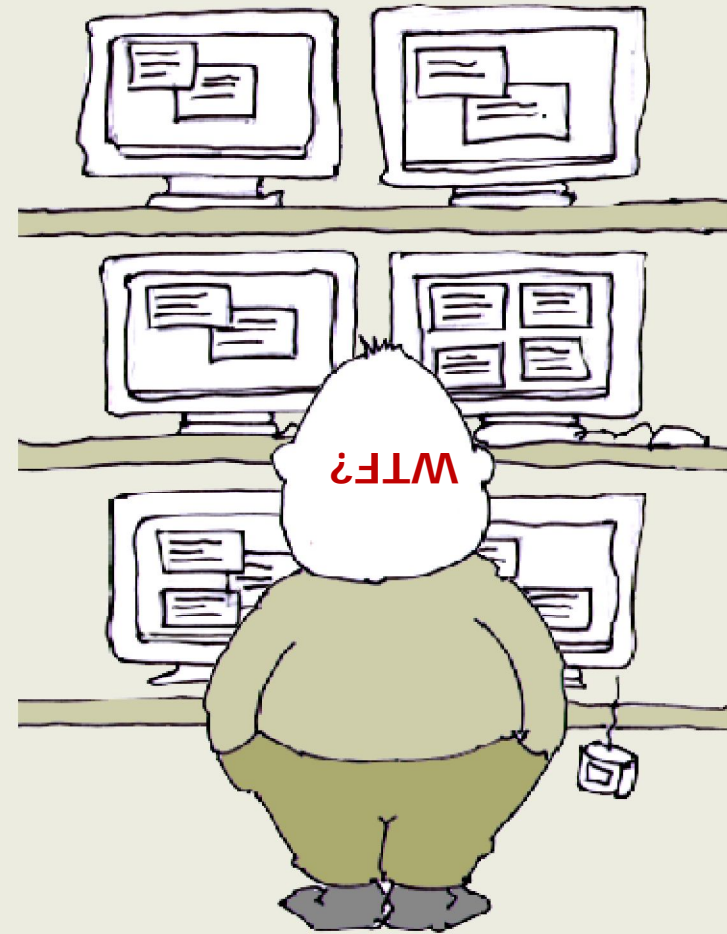
“...But TSA said it passed?!?..”

\* Edsger W. Dijkstra

# Enterprise tooling needed to support Scale?



'I see exactly where we are'



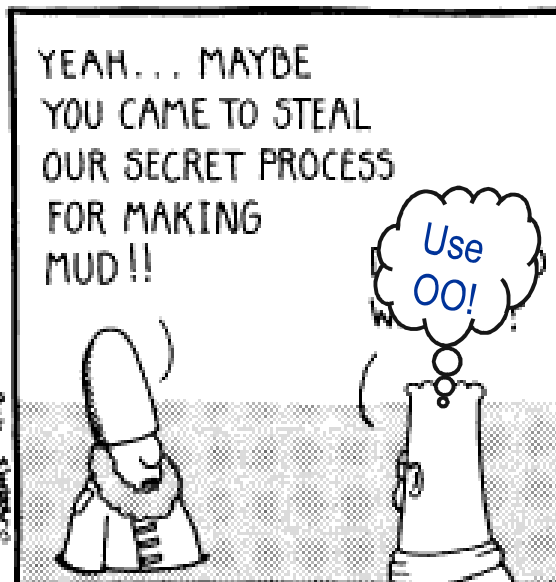
'I think we need our PM back to use this'



# Living in a Legacy – Big Ball of Mud\*

## Where does MUD come from ???

“Legacy is code where there are no tests”



DILBERT © United Feature Syndicate, Inc.  
Redistribution in whole or in part prohibited.

**KLOCS KILL!**  
**KLOCS KILL!**  
**KLOCS KILL!**

\* Universal design pattern discovered by Brian Foote ..

© 2011 Bedarra Research Labs. All rights reserved.



# OMG! It is our mess.. ahh technical debt)!?

Our framework adds so much value



Just add a little glue code and some SOA wrappers



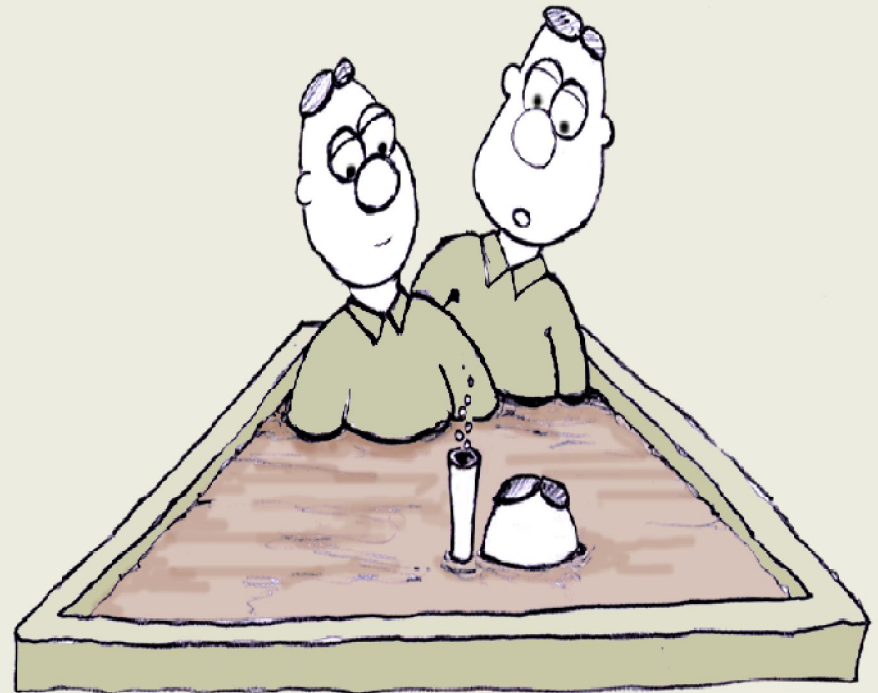
Look we will just have to refactor it later!



## Ok Refactorman tryTHIS !

Do you believe in  
Magic?

- **Wizard's** work
- **Absence of tests** significantly **increases risk**
- Tools are **not** up to refactoring **legacy** code bases or **databases**
- Off the shelf tools may **not be available** for your **legacy** language



"I don't know about you... but I find these 'legacy' refactoring tools leave a lot to be desired...  
Yeah, Let's try some dynamite"



# Estimates in the Real World

## Irritating Questions The Business Needs Answered!

How **much** will it **cost** in \$?

How **many** person **months** will it take?

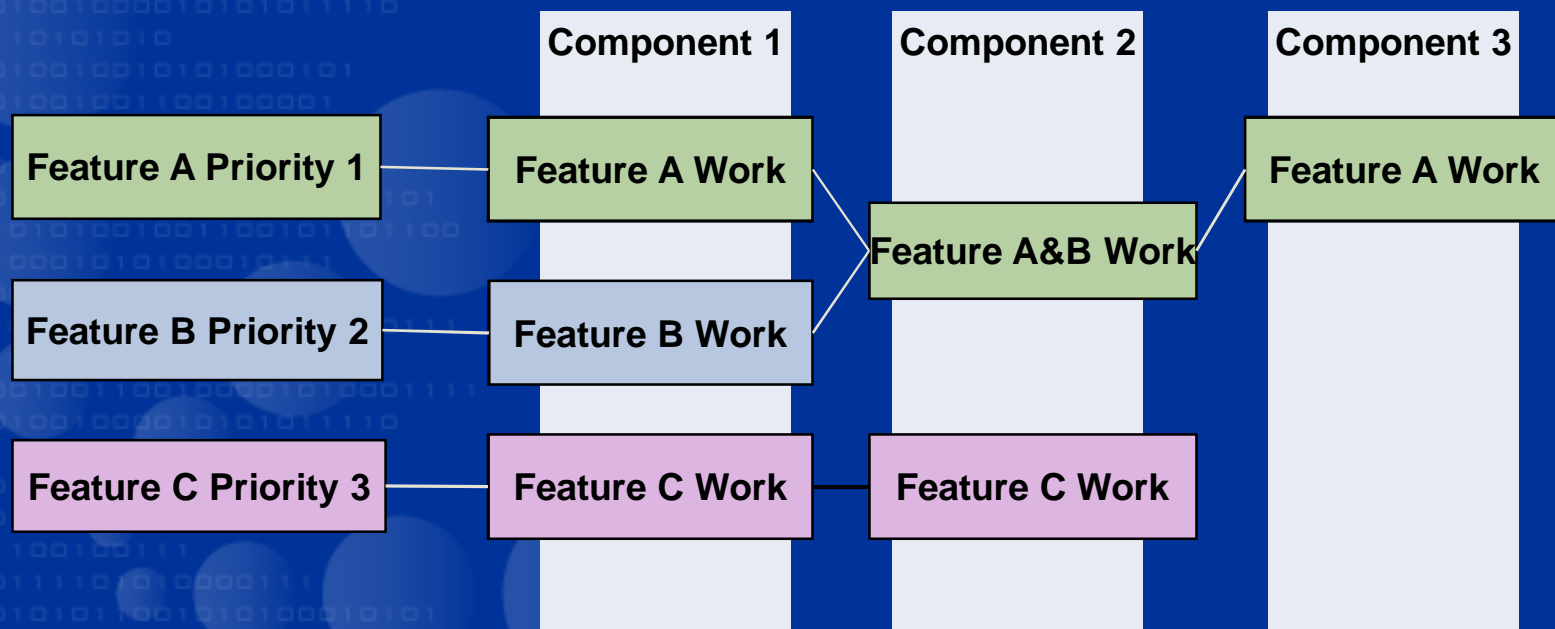
What **date** will it **ship**?

What do you mean that **your** estimates **changed**?



# Large Scale Products need Feature and Component Teams An Essential Tension

- Features sustain the business
- Components (Services) sustain the code base
- Each team owns its code and tests
- Feature backlogs are ranked on business value
- Component backlogs are ranked on feature value





**Thank You!**

# Application Development 20xx



# Challenges Ahead

- IT is perceived to be an **expense** rather than adding **value**
- **Software** is way **too hard** and there is way **too much** of it
- **Skills** are in **short supply**



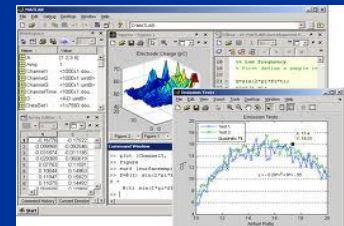
# Some Potential Solutions ...

## Education and Empowerment

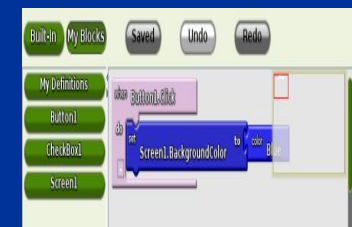
- Teach **teamwork** and **story telling** in K-6



- Invest in **non CS** Education beyond **trivial** literacy e.g. data and computational science



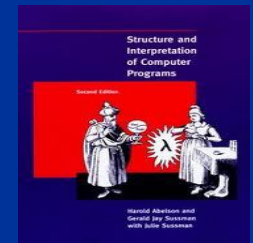
- Enable more **end user** programming



# Some Potential Solutions ...

## Change the way we build software

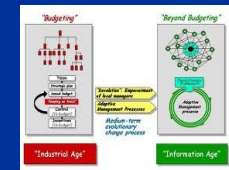
- Use more expressive **higher level** languages
- **Smaller programs** which are **loosely coupled**
- **Design for change**



# Some Potential Solutions ...

## Adopt Better Business Practices

- Business **integrated** ultimate pairs versus **isolated developer** pairs
- Sensible **budgeting** and **portfolio** management





**Thank You!**