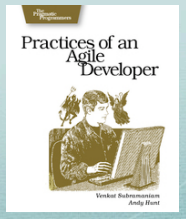
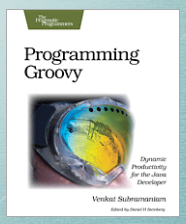
TOWARDS AN EVOLUTIONARY DESIGN

```
speaker.identity {
  name        'Venkat Subramaniam'
  company     'Agile Developer, Inc.'
  credentials 'Programmer', 'Author', 'Trainer'
  blog        'http://agiledeveloper.com/blog'
  email       'venkats@agiledeveloper.com'
}
```





Abstract

- A good design is critical for success with agile development. That does not mean a big up-front design. The design has to be evolutionary. However, the design you evolve must be extensible and maintainable. After all, you can't be agile if your design sucks. In this presentation, we will address what evolutionary design is, and will delve into principles and practices that can help realize an effective evolutionary design.

Some Myths About Agility

- Agile means fast
- Agile means ready, fire, aim
- Agile means no documentation
- Agile means no design

What's Agility?

- An approach to developing **relevant** *working* software

Agility vs. Fragility

- If you ignore design, you'll end up with fragility
- Your application breaks easily
- A small change in requirement results in massing change to design and hence code
- You begin to resist change in this case
- Hence you'll end up resisting agility

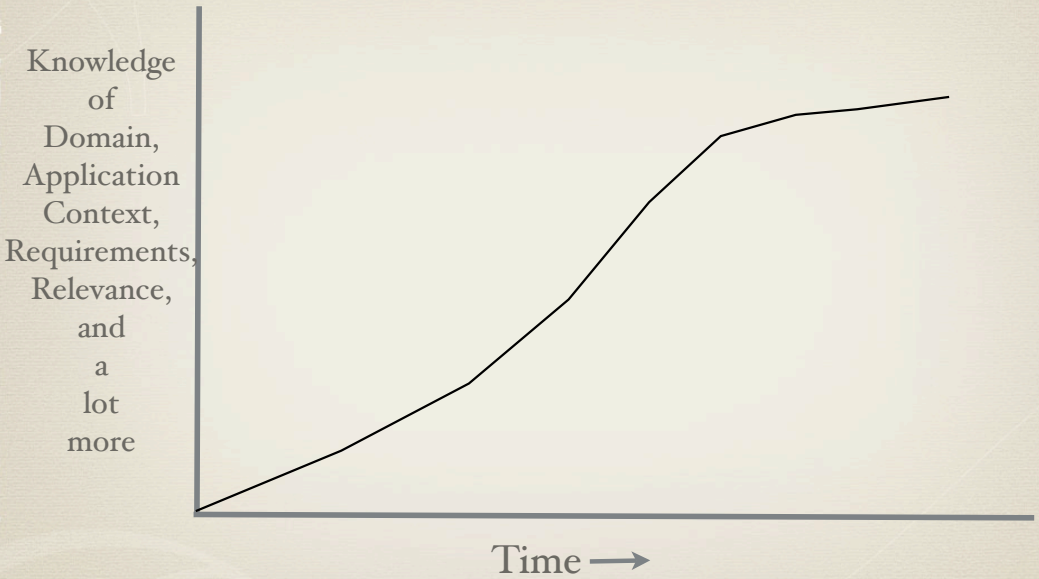
Agile Means No Design?

- Agile does not mean no design
- Agile discourages detailed up-front design
- How to approach design?

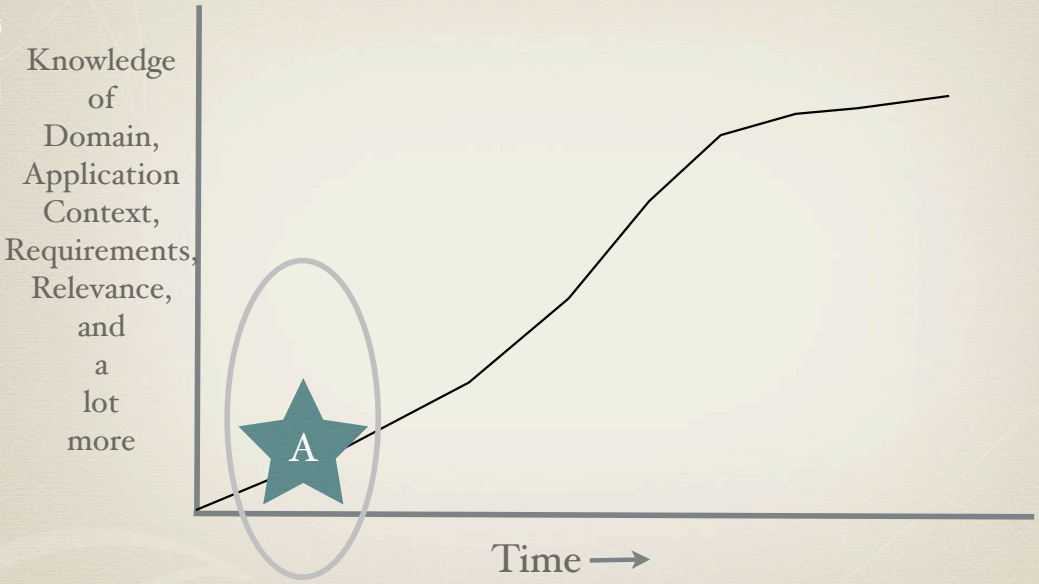
Architecture

- Very significant
- Need to get it right
- When do you typically develop Architecture?

What we Know?



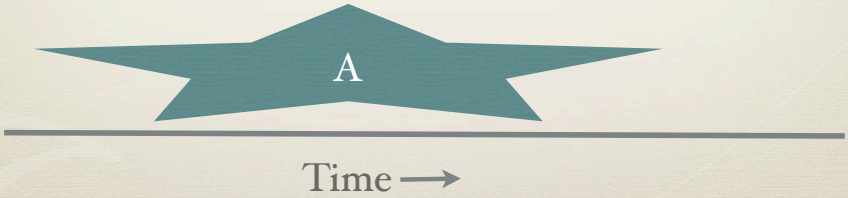
Visit that Again



One word that describes this: RISK

Why Evolutionary Design?

- Why would you take on something that important when you know the least?
- You don't want to get it wrong—so don't get it when you don't have a clue



How to Approach Design?

- Ask what are you designing?
- Ask why are you designing it?

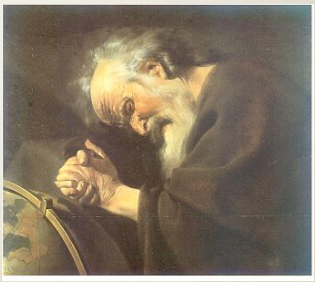
"Are you developing the software right; are you developing the right software"

What's Your Application?

- It is not easy to understand your application requirements

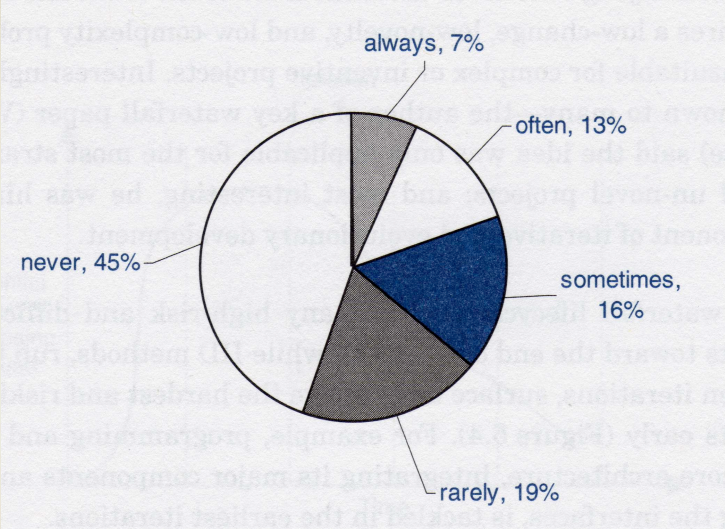
Software exhibits "Heisenberg effect - delivering software changes user's perception"—Dave Thomas and Andy Hunt, Pragmatic Programmers.

"The only constant is change"—Heraclitus.



"Walking on water and developing software from a specification are easy if both are frozen,"-Edward V. Berard.

Relevance



Actual Use of Requested Features

From Agile and Iterative Development: A Managers Guide by Craig Larman

Complexity vs. Capability

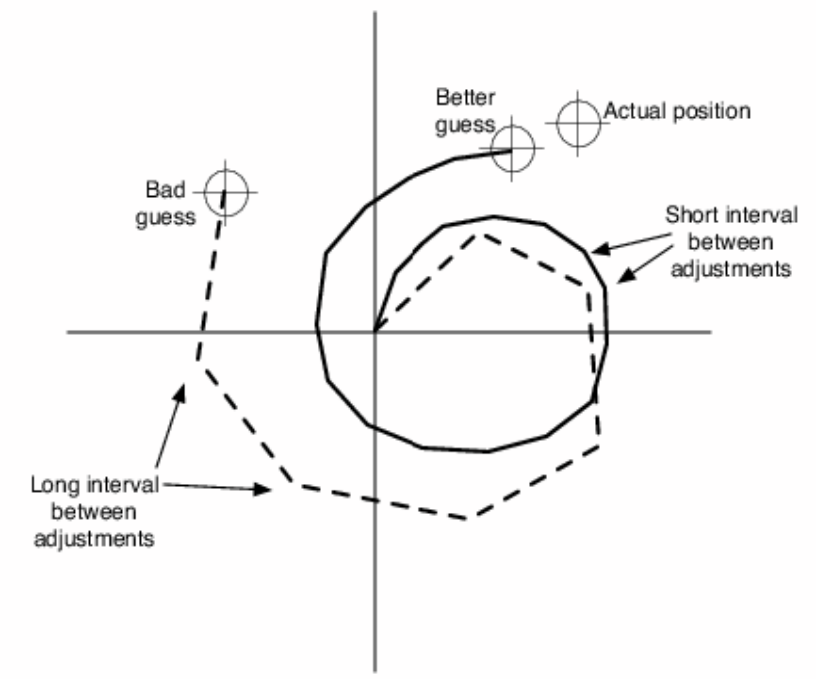
- You may have heard someone say: "I work on a large application—over 3 million lines of code"
- What does that really mean?

Complexity vs. Capability

- What is the real capability of the application?
- Who's using it?
- What are they doing with it?
- Useful, relevant features list

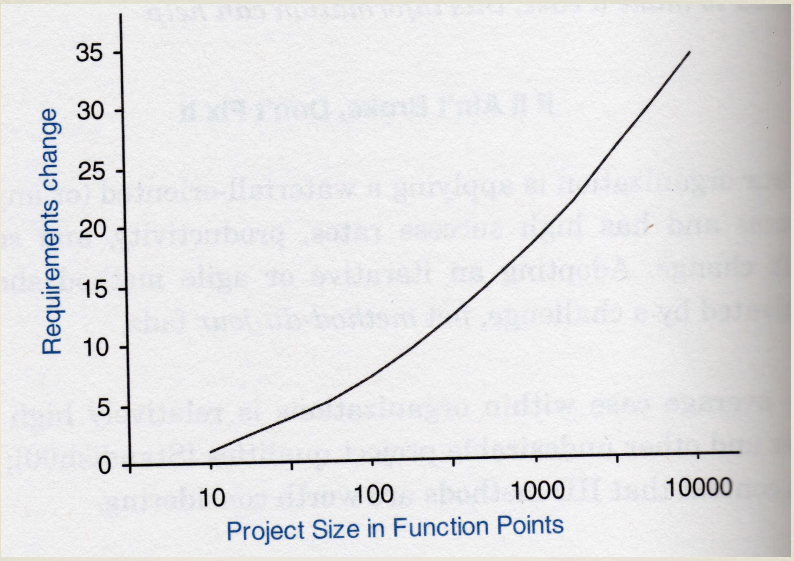
- Don't create complexity, create capability

How to learn what's Right?



From "Practices of an Agile Developer"
by Venkat Subramaniam and Andy Hunt

Change in Requirements



From Agile and Iterative Development: A Managers Guide
by Craig Larman

Plan to throw away

"When designing a new kind of system, a team will design a throw-away system (whether it intends to or not)"—
Frederick P. Brooks, Jr. in The Mythical Man-Month.

Don't Over–engineer

- It is very hard to predict all the requirements—both imminent and long term
- You want to be able to evolve your app as you get a better understanding
- KISS principle, avoid unnecessary complexity
- Parsimony—less is better—principle

Take a look at "When good-enough software is best," Edward Yourdon, IEEE Software, 1995.

Agility and Design

- It is not Agility vs. Design
- Agile does not mean No Design
- Avoid Detailed up-front Design, approach in phases
- Strategic Design and Tactical Design
- Strategic Design is high level initial design—brainstorming, modeling, ...
- Tactical Design is detailed, fine grained—TDD, collaborative,...

"The designer of a new kind of system must participate fully in the implementation"—Donald E. Knuth.

Read about "Who Needs an Architect?" by Martin Fowler.

Agility and Design

- Design is alive and well in Agile Design

"...when in doubt err on the side of simplicity. Also be ready to simplify your architecture as soon as you see that part of the architecture isn't adding anything"—Martin Fowler.

- Read about "Is Design Dead?" by Martin Fowler.

Which of these two conveys good design?

Why?

Layering

Approaching Design

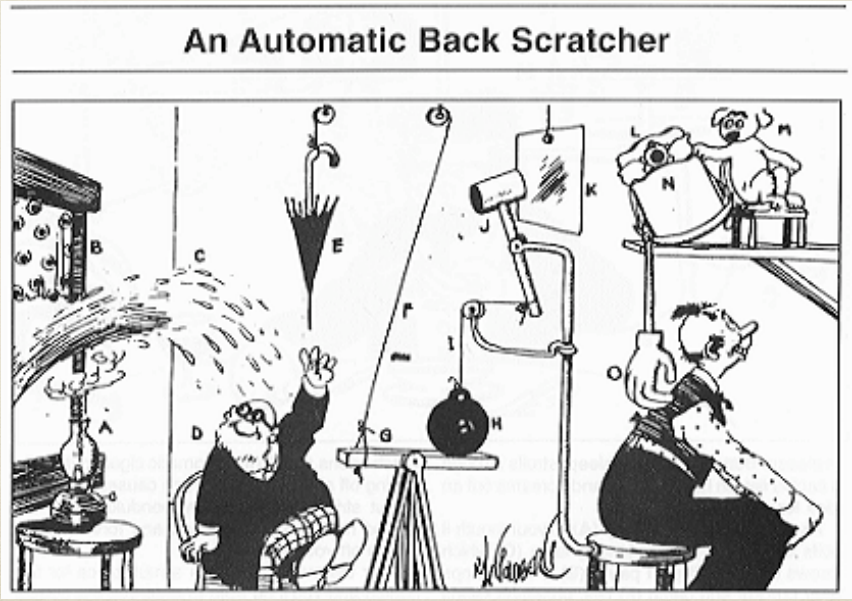
- You can do several things to help evolutionary design

Keep It Simple

- Find Simple solution that works

Keep It Sweet & Simple!

- Don't build Rube Goldberg Machines–something complex to do simple things



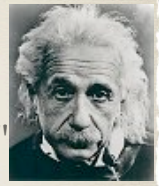
Simple!

- Which web site you visit the most?

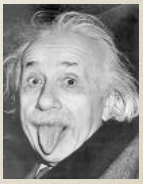
Simple, not Simplistic

- Simple is not simplistic

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius -- and a lot of courage -- to move in the opposite direction."



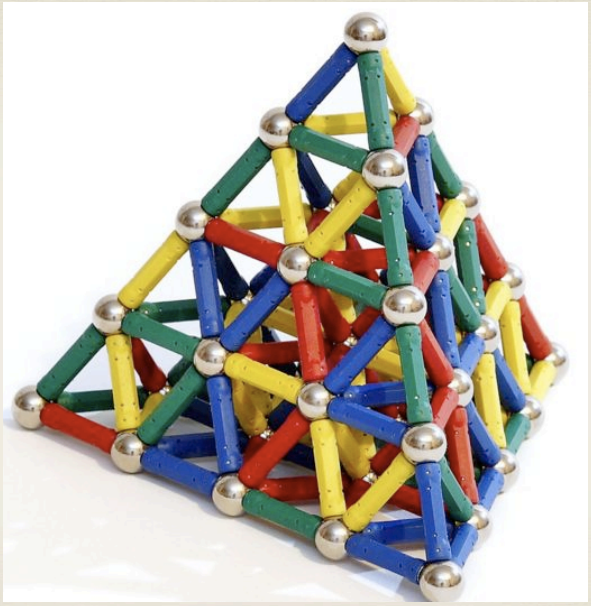
"Make everything as simple as possible, but not simpler."



Keep It Small

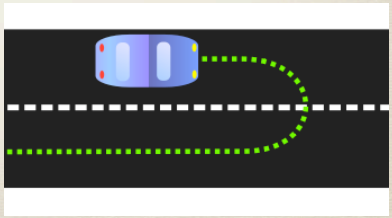
- Small makes it
  - Easy to understand
  - Easy to maintain
  - Cohesive
  - Less Coupling
  - Testable
  - More reusable
  - Easier to evolve

Small Pieces Loosely Coupled



Consider Reversibility

- Don't reach a Point of no return
- Can you back out of design decision?
- Are there things that you can't change
- What is the impact?
- Cost vs. benefit

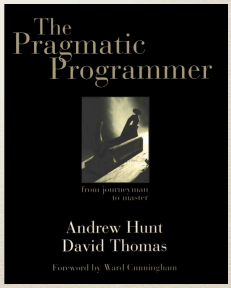


Prototype

- Creating software is highly innovative
- You try ideas, concepts, APIs, algorithms, etc.
- You don't want to endure through your real code for these
- That will limit your productivity, and you are too worried
- Prototype to experiment, learn, spike, ...
- Try it, play with it, throw it away

Keep it DRY

- Duplication of effort lowers productivity, increases cost
- Eliminate not only duplication of code, also duplication of effort
- Don't Repeat Yourself (DRY): Every Piece of Knowledge must have a single authoritative source



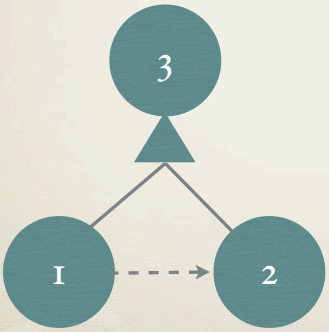
Unnecessary Complexity

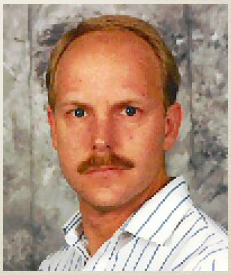
- We build abstraction, layers, interfaces, ... for the sake of extensibility or perceived functional and non-functional requirements
- Do we really need it?
- Can you postpone implementing it?
- How soon do you need that feature?
- What's cost of adding it now vs. later?

- Ron Jeffries coined the YAGNI principle: You Aren't Gonna Need It

But, What about Extensibility?

- Extensibility is very important
- But, do you know what you're extending it for?
- Write minimum code, abstract as commonality arises
- Kent Beck recommends Triangulation





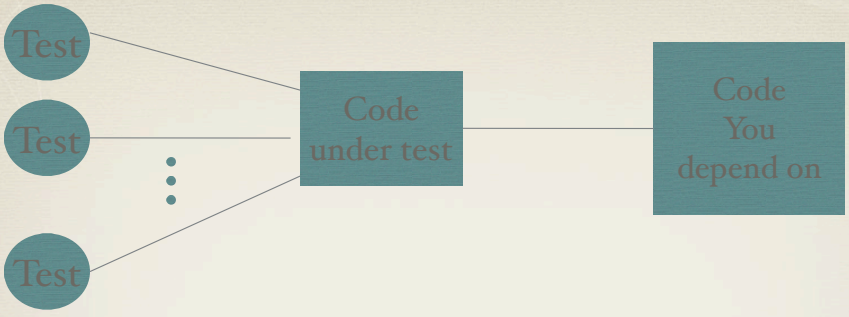
Frameworks?

- Which Framework should you use?
- Think of need, reversibility, productivity, ...
- Need should be the deciding factor
  - Need determines technology, not emotions, desires, marketing,...
  - Avoid RDD—Resume Driven Design
- Don't look at feature list of framework
- Look at feature of your application

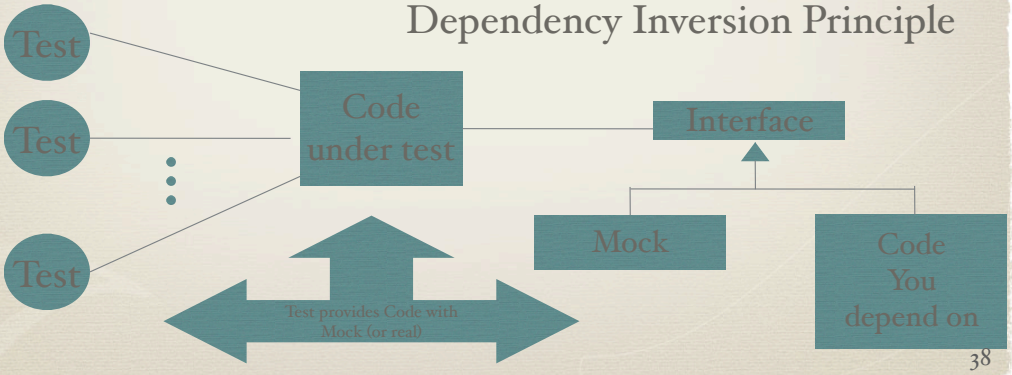
Dependency Inversion

- Strong coupling leads to a crippled system
- Loose Coupling leads to more extensible and maintainable code
- Helps with testability as well
- Depending on a concrete class leads to tight coupling, interface provides loose coupling
- Inversion of control or dependency inversion principle serves as invaluable design principle

Dependency Inversion



Dependency Inversion Principle



Test Driven Design

- How do you test a large method with tight coupling?
- Poor design is hard to test
- Small methods (cohesive) with loose coupling is easier to test
- Better design is a collateral advantage of testing
- If a code is throw away (prototype, spiking), no need to test it
- If it is useful code, you need automated tests on it

Refactoring

- "A process of improving the design of existing code"
- You're not changing the behavior of the code, you're improving its internal structure
- Why?
  - Easy to understand
  - DRY
  - Simplify
  - Readable, ...

Make it work, then make it right

Architecting

- Practice Evolutionary Architecture
- Test out ideas
- Continuously evolve, integrate, and test
- But, how

Lessons from Gunnery

- What if you are shooting at a moving target, under varying weather conditions, ...
- You can do precise calculations and take your best shot
- Or you can see and alter your angle, direction, etc.
- Every few bullets contain special chemicals that glow upon firing—these are called tracer bullets

Tracer Bullets

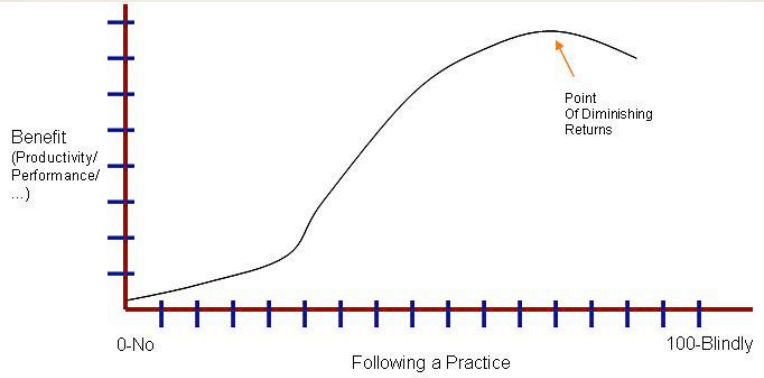
- Tracer bullets glow when fired, helping you adjust your aim as you fire
- Tracer bullet development allows you to adjust your process as you develop your application
- Any process must
  - allow inclusion of outside practices that work well
  - allow for constant reevaluation and adjustment
- A good process is the one
  - that works for you and is sustainable

Tracer Bullets

- No one shoe fits all
- Allow for experimentation
- Create an end-to-end system with hollow components so you can get a feel for the system very quickly
  - Use mocks that can be replaced later
  - Use canned data that quickly return expected results for testing
  - Don't try to perfect things right in the beginning
  - Make things look like they actually work
- Fill in real logic into this framework as you go along

Be Agile about Agile

- Any methodology or practice that becomes prescriptive and dogmatic will fail
- Use your judgment
- Strike a balance, know the limits, keep an eye on it



Thank You!

You can download examples and slides from
http://www.agiledeveloper.com - download