



Perils and Pitfalls of Agile Adoption

Carl Erickson, PhD

Premise of this talk

Agile practices, effectively and properly applied, will improve your development process. Knowing the likely pitfalls you'll run into will help.

[start here](#)



Premise:

agile practices, genuinely and correctly applied, will improve your development process

Sources:

Atomic Object – 20 developers, 5 years old, XP practices from 2000

Consulting – larger companies, variety of domains

Conferences – XPU, XPAU, Agile International in particular

Smart People – Ron Jeffries, Bob Martin, Brian Marick, Bret Pettichord, Michael Bolton, Scott Ambler, to name a few

Sources

Atomic Object - 5 years, hundreds of projects

The Atoms of AO - Bill, Dave, Micah, Drew, Dustin, Greg, Justin, Karlin, Mark, Matt, Mike, Mike, Patrick, Patrick, Scott, Shawn

Consulting - larger companies, variety of domains

Conferences - XPU, XPAU, Agile International, AWTA in particular

Smart People - Ron Jeffries, Bob Martin, Brian Marick, Bret Pettichord, Michael Bolton, Scott Ambler, to name a few

Inspiration

Matt Heusser sold the idea of this talk to SD Best Practices 2006

The material and the slides are my own

Background

Assume you know something about agile practices

[next](#)

Navigation

The perils and pitfalls described in this talk are organized into 14 top level sections. Each section has a list of hyperlinks to the related pitfalls.

Each page has a link in the top left corner that returns you to the previous organizational level. Some pages have a sequential link in the lower right corner.

Rules of the game (if you were in Boston)

You will determine what we talk about, what slides we visit

You accepted a ball from me at the start.

This ball represents an obligation to choose a pitfall.

Throw your ball at the front to make a choice.

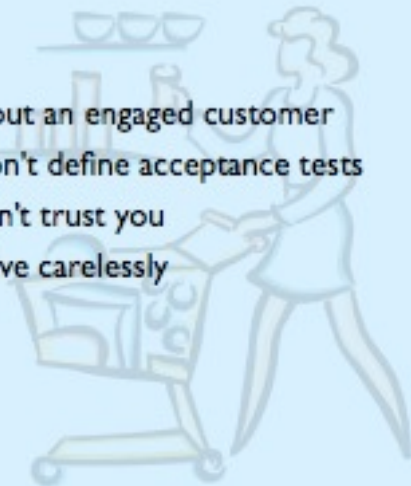
Please take the ball home.

[next](#)



Customers

- Working without an engaged customer
- Customers won't define acceptance tests
- Customers don't trust you
- Customers drive carelessly



Copyright 2008 Atomic Object LLC

Working without an engaged customer

- Being a good customer: hard, time-consuming
- You'll be missing: requirements, acceptance criteria, priorities, domain expertise
- Developers: consummate problem solvers
- Key question to ask: "Is it worth building?"

Copyright 2008 Atomic Object LLC



Customers won't define acceptance tests

What will you build?

How will you know when you're done?

Suggestion: don't use the word "test"

Copyright 2008 Atomic Object LLC

Asking them for examples, help them sketch things,
describe scenarios
whiteboards, paper, document
Question: Do you keep these artifacts?



Customers don't trust you

Side effects include:

- asking for the moon
- unwillingness to prioritize
- difficulty in phasing
- no minimal working system

Suggestion: start small, deliver early, deliver often

Copyright 2008 Atomic Object LLC



Customers drive carelessly

Customers are the driver, developers are the car

With a powerful car comes a heavy responsibility

Bad stuff: churning, thrashing, changing directions abruptly

Not keeping your eyes on the road ahead

- customer feedback
- market research



Pairing

Budgeting 2x when your development team pairs

Assuming most developers will dislike pairing

Letting the stronger person drive

Eliminating pairing, not mitigating risk in other ways

Only pairing when introducing new team members

Estimating in pairs hours

Monitors in a corner

Not pairing pragmatically



Budgeting 2x when your development team pairs

Oddly common mistake

Usually results in “no pairing” decree

Suggestion: refer to these [studies](#)

Copyright 2006 Atomic Object LLC



Pair Programming Studies

- Cockburn & Williams
 - 15% overhead for pairing (controlled, academic)
 - Improved design, defect rate, morale
- Jensen
 - “two person programming teams”
 - 1975 study of Fortran project, 50k LOC
 - Productivity 2.2x greater in pairs (LOC/person-month)
 - 1000x reduction in defect rate



Assuming most developers will dislike pairing

Poll: if you have not tried pairing, do you think you'd like it?

Copyright 2008 Atomic Object LLC

90% do



Letting the stronger person drive

Two roles: driver, navigator

A strong person driving must be careful

Suggestion: weaker partner drives, or switch frequently

Copyright 2008 Atomic Object LLC


Helps to have 2 keyboard, 2 mice

design reviews that aren't done seriously

 **Eliminating pairing, not mitigating risk in other ways**

- Single points of knowledge
- Complexity, opaqueness, and over-design
- Opportunities to be mentored, learn
- Wasting time being stuck
- Not following standards, best practices
- Increased developer fear ("my pair has my back")

Copyright 2008 Atomic Object LLC

 **Only pairing when introducing new team members**

- Training and ramping-up is obviously beneficial
- Reverting to the "2x pitfall"

Copyright 2008 Atomic Object LLC

Estimating in pairs hours

Developer: "that will take 10 pair hours"

Customer: "so about \$1000"

Developer: "no, about \$2000"

Customer: "I can't afford pairing!"

Suggestion: estimate work for pairs, multiply by 2, report plain old hours to the customer

Monitors in a corner



Not pairing pragmatically

Solo work is ok when

- There's an odd number of developers
- You have an experienced person
- You have "cloning" work to do
- You have exploration/learning to do

No compromise on

- All new code
- All design questions
- All testing challenges

Testing

Writing fat unit tests

But you can't test X!

Thinking about TDD as testing

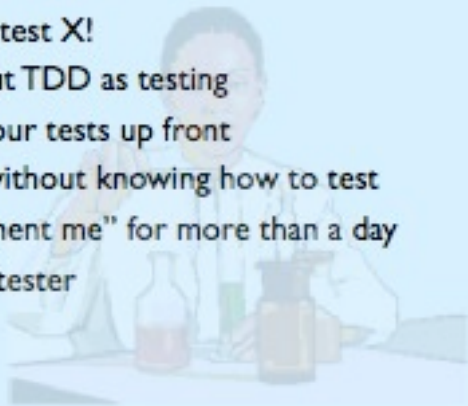
Defining all your tests up front

Doing TDD without knowing how to test

Using "implement me" for more than a day

Fair weather tester

Legacy code



Writing fat unit tests

Very common TDD beginner mistake

State-based testing contributes to bloat for integration tests

Impacts: suite run time, refactoring cost

Suggestion: learn to use interaction testing

Copyright 2008 Atomic Object LLC

Easier to inflate a unit test to a system test than keep it focussed.

The integration tests (larger, more complicated, more objects) are the ones particularly prone to be fat.

Interaction-style testing limits the boundaries of these test.

Interaction Testing

Integration tests often involve multiple objects

Doing state-based assertions makes for fat tests

Mocking neighboring objects keeps integration tests lean

Plus: interaction testing is a means of discovering needed responsibilities (a design activity)

"Mock Roles, not Objects", Freeman, Pryce, Mackinnon, Walnes, OOPSLA 2004

Copyright 2008 Atomic Object LLC



But you can't test X!

Where X = {embedded, stored proc, function, system call, report, GUI, legacy code}

Unlikely, but if not, you've got a bad design

Suggestion: look at testing as just another problem to solve, consider changing the design



Thinking about TDD as testing

Misses out on the many non-bug finding advantages

Suggestion: don't do it

Why TDD?

- Just-in-time specification
- Catalyst for communication (pairs)
- Documentation of behavior
- The first client of a module
- Supports collective code ownership
- Continuous code improvement
- Better design (looser coupling)
- Pace of development is smoother
- Avoiding technical debt

Copyright 2006 Atomic Object LLC



Who, when, why?

Automated testing...
is done by developers,
while they write source code,
to know when they are done,
to document what they have done,
to extend and maintain code fearlessly.

Copyright 2006 Atomic Object LLC



Thought experiment:

A colleague asks you to build some code that performs in a certain way.

What do you do when you develop the new method or function?

Do you just code it up and hand it to them?

Do you compile it first?

Do you run it a few times?

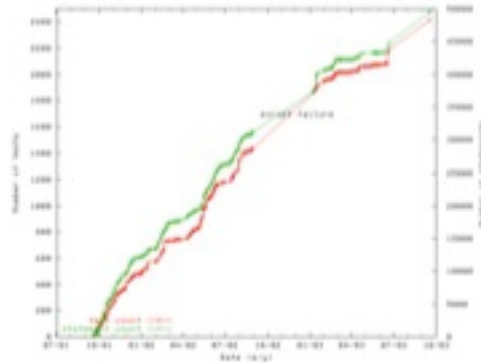
How do you know it works?

You test a few interesting cases.

You might have to write a little jig to hold your test.

What happens to the testing code, typically?

TDD and Deadlines



Copyright 2008 Atomic Object LLC

Testing can't be eliminated in a pinch
So technical debt can't build
so velocity doesn't slow
so you can meet your deadlines
so your company can stay

competitive

so you can keep your job

Defining all your tests up front

The BUFD approach to TDD

You'll write code you don't need

It may be days or weeks before you see green bar

Suggestion: don't do it

Copyright 2008 Atomic Object LLC



Doing TDD without knowing how to test

Good tests are A-TRIP

Use these mnemonics

Comes with time and experience

Suggestion: find an experienced test-infected developer

Copyright 2008 Azamir Object LLC

A little testing education goes a long way
Don't get bogged down in the analytic school



Characteristics of good tests

A - automated

T - thorough

R - repeatable

I - independent

P - professional

Copyright 2008 Azamir Object LLC

Right BICEP

- Right** stuff is computed? Results as expected? How would you know? (test that)
- Boundary** conditions handled correctly? (think about equivalence classes)
- Inverse** relationship works? (e.g. check that square of square root is original number)
- Cross-check** results some other way? (perform the operation some other way and check)
- Error** conditions correct? (force errors, confirm exceptions, expected error return, etc)
- Performance** characteristics ok? (to spec, or?)



Copyright 2008 Atomic Object LLC

CORRECT

- Conformance** - to proper format?
- Ordering** - ordered or unordered as hoped?
- Range** - within range?
- Reference** - what does the code depend on?
- Existence** - non-null, etc
- Cardinality** - number of values right?
- Time** - in order? right time? on time?

Copyright 2008 Atomic Object LLC



Using “implement me” for more than a day

The test you recognize needs to be written

```
fail("implement me");
```

People stop expecting the green bar

auto plants: \$100,000 / hour downtime penalty



Fair weather tester

“It’s due Tuesday!”

“Don’t shut the plant down!”

“We don’t have anyone to pair you with.”

“You’ve only got 40 hours!”

Suggestion: test infected developers

Legacy code

Agile practices won't magically undo years of technical debt

Suggestion: seriously consider dumping it

Suggestion: integration/system tests help you define expectations

Suggestion: don't let the old pollute the new

Suggestion: slowly carve it out and replace it

Copyright 2008 Atomic Object LLC

dumping it:

you've got a working system to test against
the value isn't as much the code per-se, as the
knowledge it encompasses

you've got pretty good requirements
you may be able to take advantage of new
technologies, practices, tools

Professional Responsibility

Hiding the truth

No brave people

Not taking personal responsibility



Copyright 2008 Atomic Object LLC



Hiding the truth

Common when asked

- for an estimate
- to accept a date
- whether something is done

Suggestion: practice speaking truth to power

Copyright 2008 Atomic Object LLC

Doesn't do anybody any good

Fears about evaluations, raises, job security are often overblown

If they aren't, do you really want to work there?



No brave people

Related to the pitfall of hiding the truth

Effective agile developers

- care deeply about quality and production
- are passionate about their profession

Are therefore willing to

- point out that an artifact is useless
- arm themselves with wrenches and screwdrivers
- learn new tools, technologies, techniques
- engage in a job-risking fashion

Copyright 2008 Atomic Object LLC

Courage is one of the four XP principles



Not taking personal responsibility

Producing code you can't prove works

Accepting unrealistic estimates or deadlines

For quality, broadly defined

For all aspects of software development



Advanced

Thinking you're immune to project bit rot

Stopping at state-based testing

Estimating testing and development separately

Thinking that automated unit testing is enough

Not testing the environment

Agile developers need company

Misunderstanding system tests



Thinking you're immune to project bit rot

You've been doing the basics for a while

- automated testing
- customer prioritized, development
- iterations

What happens to projects in maintenance?

Suggestion: continuous, automatic build + visibility



Continuous Integration

CI = automatic build + test

DCI Monitor

Thu Jul 09 10:04:12 EDT 2008

rails	svn	heroku
rails2	akridis	dci_monitor
s13server	cycle	rails_monitor_app
rails_dev_updates	victor	percepsion
retomart	purban	bitbucket_rails2
rails2	herc	svn
only	speedcontrol	

the biggie: Cruise Control
roll-your-own: DCI in Ruby, DCI Monitor



Stopping at state-based testing

State-based: invoke unit, assert on state

- simple, core practice
- harder for integration tests
- increases test maintenance

Interaction-based: mock neighbors, assert on interaction

- correct method called
- ordering of methods
- good tools available

Suggestion: learn how to use both techniques

Copyright 2008 Atomic Object LLC

spectrum of testing from unit -> integration -> system
automated unit + integration tests generally in same suite
interaction testing makes for focussed, tight integration tests, just like unit tests



Estimating testing and development separately

Would you hand a kid a loaded gun with the safety off?

Remember, the T in TDD isn't really "testing"

If you're willing to let customers eliminate testing...

Copyright 2008 Atomic Object LLC

Customers are usually less versed in your craft than you are.

You don't tell them what features to put in their app...

Helps to reduce cost of testing

Story: AO and unit->system testing



Thinking that automated unit testing is enough

Reducing bugs that hit production by 10x

The missing piece: exploratory testing

Suggestion: makes friends with a good exploratory tester

Copyright 2008 Atomic Object LLC

Pretty amazing to reduce bugs by 10x
Unless you are a very big team you won't keep one busy
all the time

Story: AO and first large project

Exploratory testing is more than finding bugs: usability,
configuration, compatibility with previous versions,
installation

Story: AO and customer trust - important demos with no
prior manual testing



Not testing the environment

Software is designed to run in a particular situation

- operating system
- authentication/authorization
- environment

Fault isolation can be costly

Suggestion: automated tests encode assumptions

Copyright 2008 Atomic Object LLC

AO examples: kiosk, order entry app, web apps and
libraries

Create tests that define the assumptions you made about
the environment while building the software



Agile developers need company

Don't expect a single agile seed to grow in a traditional garden

Suggestion: pairs are powerful

Copyright 2008 Atomic Object LLC

Story: AO as contractor, customer desire to spread practices, work at customer location



Misunderstanding system tests

TDD doesn't leave many bugs to find (10x reduction)

Important role: integration, build, and regression

Leaving them for last, not automating is a bad idea

Suggestion: drive development with system tests

Copyright 2008 Atomic Object LLC

Finding bugs with system tests is horribly inefficient
Story-driven development keeps developers focussed on customer priorities
Periodically coming back up for guidance after a deep dive into code



Leadership / Management

Individual metrics, rewards, evaluations
Not trusting your team
Specialization
Confusing roles and responsibilities
Moving on without celebrating
Failure to inspire
Middle management resistance
Making bogeymen of external forces
Ignoring bad apples
Lack of executive support

Copyright 2008 Atomic Object LLC



Individual metrics, rewards, evaluations

Agile is a team sport

- co-located, tightly-coupled
- sharing, helping
- team responsibility

Teasing out individual contributions is hard, and potentially counter-productive

The corporate "third rail"

Suggestion: do some research

Copyright 2008 Atomic Object LLC

Agile teams often pair, they usually take team responsibility for tasks

Story: maintenance team, change-controls-per-month by person

3rd rail: Distinct, individual compensation

Agile Metrics, Compensation

"Six Dangerous Myths about Pay", Jeffrey Pfeffer,
Harvard Business Review, May/June 1998

"Appropriate Agile Metrics: Knowing What and
When to Measure", Hartmann, Dymond,
Agile International Conference, 2006

Copyright 2006 Atomic Object LLC

Not trusting your team

Agile developers take pride in and know their craft

Craftspeople learn new tools quickly

Craftsmanship drives process innovation

Agile practices deliver

- working software regularly
- data you can manage with

Copyright 2006 Atomic Object LLC

Developers know their craft - should you really second
guess them?
Learning new tools, technologies, languages isn't so hard
Innovation requires some room to experiment

AO on system testing:

Java GUI automation -> manual ->

organizational pattern -> ?

AO on web development

classic perl CGI

OO perl

PHP

PHP with template library

XML framework

Rails

Specialization

Role specialization causes

- interfaces between specialists
- translating between specialists (non-source artifacts)
- responsibility shifting

You don't want PhDs, you want craftspeople

Suggestion: listen to [Lazarus Long](#)

Copyright 2006 Azamit Object LLC

many roles: architect, dba, tester, analyst, programmer, proj manager

The business analyst tries to express requirements in English. They are

ambiguous, incomplete, expensive to produce, often wrong

The architect tries to express an architecture with diagrams. They are

usually created at the wrong time, a long way from design or code

The DBA designs tables without knowing how the application will use them

Lazarus Long on Specialization

A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. **Specialization is for insects.**

Robert Heinlein,
Time Enough for Love

Copyright 2006 Azamit Object LLC

Confusing roles and responsibilities

Team is the car

instruments, fuel efficiency, turning radius, compass

Customer is the driver

where are we going? what route shall we take? when will we get there?

Developers take responsibility for dates too readily

Suggestion: produce data for customer/manager to steer by

Copyright 2006 Azurix Object LLC

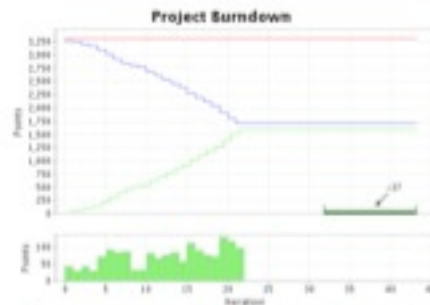
Data-driven Project Management

Ready to reporting mode

Est	Est	Weekend	Items	Remaining	%	Projected Month
11	11	11.00	1 New BPC	10	100	9
8	12	8.25	2 New requirements	24	100	9
1	9	4.25	1 Existing New External Feature	12	100	9
11	11	11.00	1 Existing Feature Release	10	100	9
1	11	1.10	1 New Feature	10	100	9
1	9	1.00	1 New Feature	10	100	9
11	11	11.10	1 New Feature	10	100	9
1	11	11.00	1 Project Management	10	100	9
11	11	11.00	1 New BPC	10	100	9

Total Features: 107
Total Work: 108
Remaining in Estimate: 17
Projected Remaining: 9
Projected Completion: 100%

Burndown Chart
Generated: December 11, 2005



Project Work Remaining Work Completed Story Points Completion Date

Copyright 2006 Azurix Object LLC

Quite distinct from plan-driven project management

Small projects: tabular report by phase

Larger projects: burndown chart by iterations



Moving on without celebrating

Agile makes meeting budgets and deadlines normal.

Applications usually just work as intended.

There's always another project to move on to.

Suggestion: make a ritual

the ultimate job of leadership



Failure to inspire

Too often focus on culture change pain

Worrying about impact on legacy roles

Technical difficulties, baggage of legacy code

Suggestion: talk about a future of integrity, quality, pride of craft, innovation, efficiency, business success



Middle management resistance

Legitimate fear: what is my role, if it's not:

- task assignment
- reporting data from team
- customer liaison

Capital One: lean + agile

22 manager peers reduce to 4

no project cancellations

30-50% faster, no reduction in quality

10-15% cost reduction

Copyright 2006 Atomic Object LLC

Managers were good at doing what the legacy system valued.

Changing that system causes legitimate fears.

Capital One: Agile Int. Conf, 2006



Making bogeymen of external forces

"But the auditors said..."

SAS70, HIPPA, CMM, EVM, DO-178B

Suggestion: seek intent, be creative, don't assume

Copyright 2006 Atomic Object LLC

Ignoring bad apples

Agile shines light on many things

- one big room
- pair programming
- automatic build
- estimating, measuring velocity

Agile doesn't solve personnel problems, but it may expose them

Copyright 2008 Atomic Object LLC

Open facilities
Pair programming
Continuous integration and build
Estimates and velocity

Cost to team and individual morale
Distraction to manager from already difficult agile adoption challenges

Lack of executive support

Running under the radar

- usually works for engineering practices

Engineering practices are self-sustaining

Most likely to hit limits of this approach with

- facilities
- customers
- legacy processes
- planning

Copyright 2008 Atomic Object LLC

Customer is part of the team

Project Management

Confusing plan the noun with plan the verb
Thinking Scrum is sufficient
Velocity without distance
Counting on a team increasing velocity
The wrong metrics

Copyright 2008 Atomic Object LLC

Confusing plan the noun with plan the verb

Project managers may think about plans, not planning
Agile methodologies plan continuously
Steering, adapting, mitigating risk, tracking, projecting
vs
Tracking conformance to “the plan”

Copyright 2008 Atomic Object LLC

Project plans: often created at the point of maximum ignorance

planning is too important to be done once

the world changes too much during the project

Thinking Scrum is sufficient

Scrum speaks to roles, iterations, and customer priorities

Scrum has nothing to say on engineering practices

Suggestion: use Scrum as the interface to the customer,
but follow the rest of XP

Copyright 2006 Azamir, Object LLC

Scrum has definitely won the marketing game (vs XP's planning game)

XP Practices



Copyright 2006 Azamir, Object LLC

Velocity without distance

Agile teams measure their development velocity

Burndown or Burnup charts turn this velocity into a prediction of completion time

Where does the top red line (distance to go) come from?

Suggestion: estimate in frequency and detail as business need justifies

Copyright 2006 Atomic Object LLC

Development team: the car
Customer: the driver
velocity is the speedometer on the car

extreme: full story decomposition and estimation
extreme: crude subsystem estimates (+- 50%)

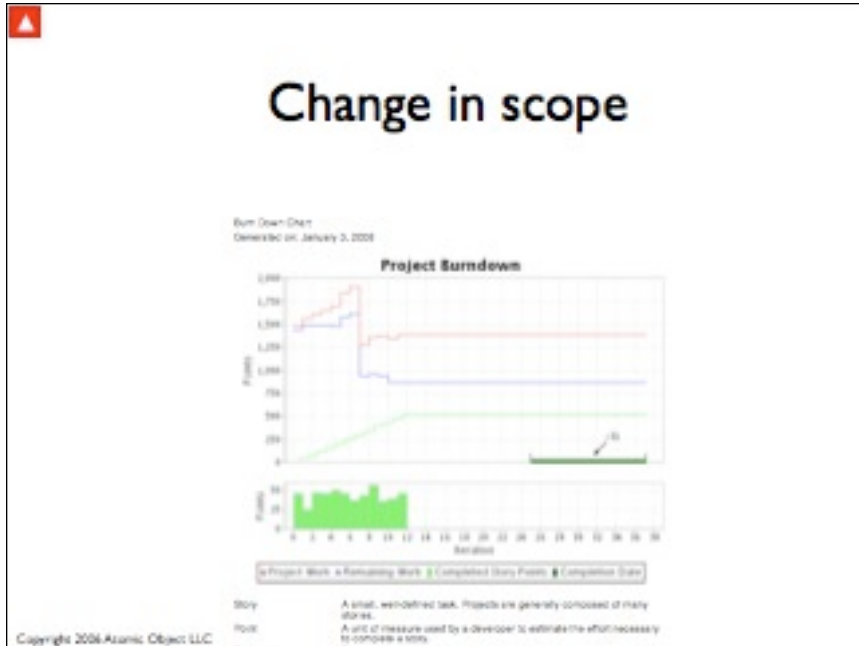
The top red line



Copyright 2006 Atomic Object LLC



top red line is the total amount of work to be done
scope creep pushes the red line up
removing features pulls the red line down
notice: this red line didn't move



top red line is the total amount of work to be done
scope creep pushes the red line up
removing features pulls the red line down
notice: this red line didn't move

Counting on a team increasing velocity

Teams take some time to establish rhythm, ritual, master technologies, gel

So it would seem reasonable to expect velocity later in the project to increase

But the refactoring burden also increases

Suggestion: either, assume it won't change much from initial, or use exponential moving average

Copyright 2008 Azamir Object LLC

We use an exponential moving average with alpha = 7/8

$$V_{\text{new}} = \alpha * V_{\text{latest}} + (1-\alpha) * V_{\text{old}}$$

The wrong metrics

Using metrics designed for traditional processes

- individual developer vs team
- discrete vs continuous (scope delivery)
- defect rate

Agile projects naturally generate valuable metrics

- test bulk, status
- story count, status
- development velocity

Copyright 2006 Atomic Object LLC

Stories about bad metrics

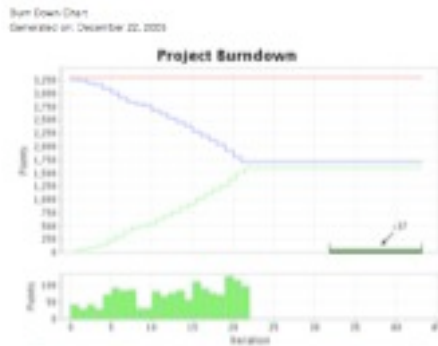
1. Maintenance team was historically measured by percentage of requests handled in a given time period.

Not requests/developer, not even total number of requests.

The team was not consulted on the denominator (requests desired to be completed)

The variance of the complexity of requests was large

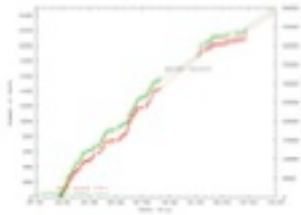
Reason for resisting change to this metric? the customer



Story	A small, well-defined task. Projects are generally composed of many stories.
Point	A unit of measure used by a developer to estimate the effort necessary to complete a story.
Project Work	The sum, in points, of all stories in the project.
Completed Story Points	The sum of the points of all stories completed during the iteration.
Remaining Work	The total project work minus sum of points for all completed stories.
Project Velocity	The exponential weighted average of completed story points per iteration. A measure of the rate at which work on the project is being completed.
Project Completion Date	The estimated completion iteration for the project. Calculated by dividing the remaining work by the project velocity.
Completion Date Uncertainty	The expected range within which the project will be completed. The variance diminishes linearly as the project moves toward completion.

Copyright 2006 Atomic Object LLC

red line is the total amount of work to be done
green line is sum of work done
blue line is remaining
scope creep pushes the red line up
removing features pulls the red line down



Name	Lines	LOC	Classes	Methods	M/C	LOC/M
Helpers	171	149	0	8	0	16
Controllers	860	686	16	95	5	6
APIs	0	0	0	0	0	0
Components	0	0	0	0	0	0
Functionals	1758	1344	35	127	3	8
Models	393	326	22	41	1	5
Units	743	559	17	58	3	7
Total	3917	3064	90	319	3	7

Code LOC: 1161 Test LOC: 1903 Code to Test Ratio: 1:1.6

.....
Finished in 62.60977 seconds.

80 tests, 687 assertions, 0 failures, 0 errors

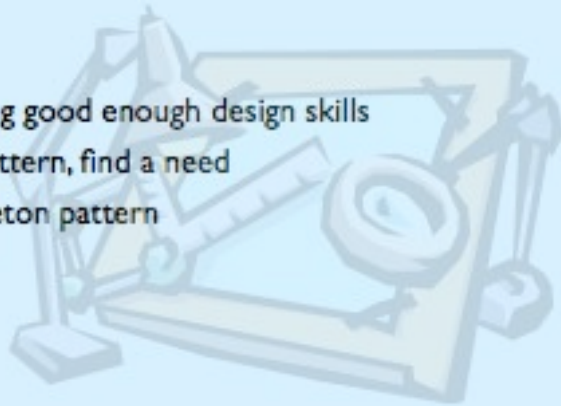


Design

Not having good enough design skills

Have a pattern, find a need

The singleton pattern





Not having good enough design skills

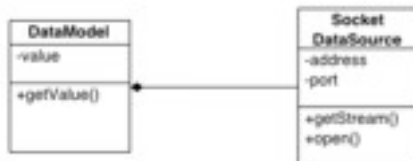
Makes refactoring more frequent, more expensive

Makes testing more difficult, expensive

Tempts you back to specialists and up-front work

Suggestion: use the power of the ever-present question

How am I going to test this?

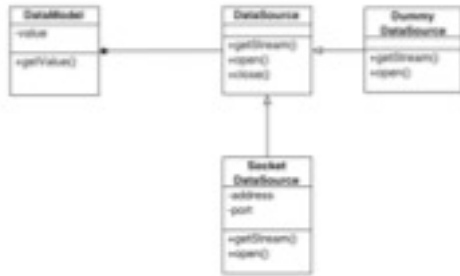


Design for testability is better design

Being pushed to answer the ever-present question will in turn push you towards understanding design principles

The single most powerful, concrete action you can take to become a better designer is to try and answer this question.

Better design



Copyright 2008 Azamir, Object LLC

Have a pattern, find a need

Aka: have a hammer, find a screw

This isn't a pitfall unique to agile development

You can push back with YANGNI or Simple Design

Suggestion: start with principles first

Copyright 2008 Azamir, Object LLC

Bob Martin's book "Agile Software Development: principles, practices, patterns"

The singleton pattern

Unfortunately easy to understand, apply, find apparent need for

Makes testing difficult

- coupling between tests methods
- difficulty in mocking

Suggestion: modify the pattern, or use a different design

Copyright 2006 Atomic Object LLC

dependency injection framework can help with composition of objects

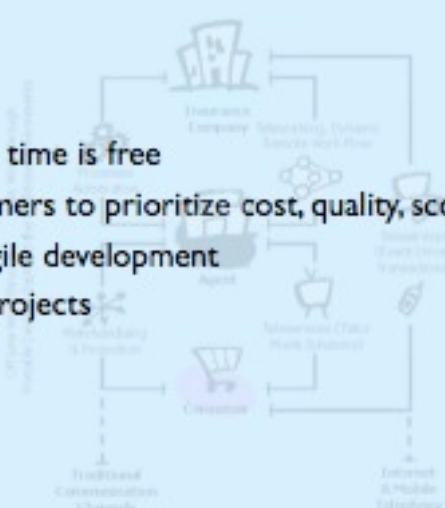
Business Model

Development time is free

Asking customers to prioritize cost, quality, scope

Fixed price agile development

Really small projects



Copyright 2006 Atomic Object LLC

Development time is free

Applies to internal development teams

If customers don't pay for dev time, they may...

- not fully engage on the team (they are busy)
- not think about value or ROI
- never stop asking for more
- not fully understand their business

Suggestion: internal chargeback? hire contractors? get better at IT governance? manage the portfolio better?

Copyright 2008 Azamir Object LLC

ironically this is an advantage of using an outside contractor

This is a hard problem. I think agile iterations and regular delivery actually helps

Asking customers to prioritize cost, quality, scope

Most customers don't understand the levers well enough

Suggestion: scope should be the only control offered

Copyright 2008 Azamir Object LLC

controlling development isn't simple

refactoring legacy code - costly in short-term, saves in long-run

building a testing framework - costly in short-term, saves in long-run

maybe the customer doesn't directly bear the long-run cost

adding people (cost) is hard to do efficiently, has team-size limits

quality is the only hope for better throughput, lower cost
Ken Schwaber's arguments about the life of a



Fixed price agile development

Customer insists on a fixed price project

The development team is committed to doing the right thing

Suggestion: Keep it small, earn their trust, migrate toward optional scope contract

Copyright 2008 Atomic Object LLC

iterations, feedback, letting the customer steer

risk for developers: adapting to change, taking feedback, letting customer steer

risk for customer: being locked into what they don't want

risk for both: wasting time arguing contracts, requirements, intent



Really small projects

Agile sweet spot: 2-4 pairs, 6-12 months

- interesting problems to solve
- commitment by customer (time and money)
- opportunity for rhythm and ritual
- big enough to fail spectacularly

Really small projects are more challenging

- financing and budget
- customer time and commitment
- start and stop pattern
- consistency of developers

Copyright 2008 Atomic Object LLC

Agile conferences the last few years: scaling agile up

My interest is the opposite: scaling agile down

Culture

Talk, talk, talk

Giving up too soon

Agile is going to fix everything

It's just words

Letting need for adaptation become a license to ignore

Underestimating the facilities problem

Change everything

Being stifled by existing culture

Copyright 2008 Atomic Object LLC

Talk, talk, talk

Talking about practices isn't the same as doing them

Favor concrete experiment and experience over talking

Suggestion: just do it



Copyright 2008 Atomic Object LLC

Story: One hour discussion in a standards group decides that interaction testing isn't valuable. Nothing concrete, vague context, no experiments, no experience.



Giving up too soon

Tools, approaches, thought processes - take time

A short trial (1 hour, 1 day, 1 week) isn't enough

Experience and a good coach can help

Suggestion: Focus on the right questions

Not: how do I do TDD?

But: how do I prove this method works?

Not: how do I do my 9 month project in iterations?

But: what single feature can I deliver or demo by Friday?

Copyright 2008 Atomic Object LLC



Agile is going to fix everything

People problems, organizational problems, technology problems, marketing problems

Suggestion: inspire somebody to worry about the "process above the process"

Copyright 2008 Atomic Object LLC

agile doesn't fix everything
it's a flashlight in a dark room
not willing to fix what it reveals?



It's just words

New agile practices and concepts can be mapped onto existing legacy terms

Continuing with old familiar terms can blunt the point of and significance of change

Words are all that we work with, words are powerful

Suggestion: be explicit about new terms, buy-in to use them



Letting need for adaptation become a license to ignore

"We thought about that [or tried, trivially] and it's not right for our organization."

"We've adapted this practice to our particular situation (just like the agile guy says)."

Suggestion: Agile is what you do after you've mastered all the practices (Ron Jeffries)



Underestimating the facilities problem

Underestimating: the impact on team interaction

Underestimating: the difficulty of changing

Owning too many desks, computers

Suggestion: carpe wrenchum

Copyright 2008 Atomic Object LLC

Story: Working on-site at our customer in typical open office plan (cubes).

Worst of both worlds: enough barrier to inhibit much technical collaboration.

Enough barrier to make people have some sense of privacy and to talk inappropriately.

Story: cardinal sin (liability of some sort) to mess with cubes

carving decent space out of the cubes



Change everything

Too much change, all at once

Too little change, slowly

Suggestion: know Satir change curve

Copyright 2008 Atomic Object LLC

test automation, iterations, stories, pairing, ...

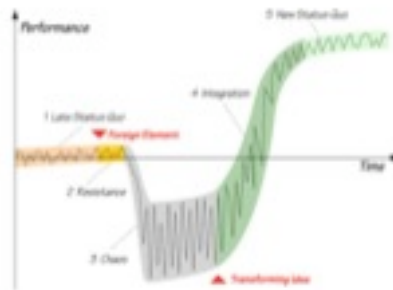
risk of incremental change

nothing much changes

new problems aren't addressed

skeptics see it as a passing fad

Satir change curve



Copyright 2008 Azami Object LLC

Being stifled by existing culture

Setbacks, expectations, visibility

Pressure to conform, morale

Suggestion: physically co-locate (perhaps offsite) to start

Copyright 2008 Azami Object LLC

AO story: easier to start from scratch than change culture

IBM PC story

Dyno host project: 6 dev, 9 months, onsite at AO, pairing AO-Bepco, 1 week iterations, transition back in last month, agile nucleus

Basement team room story: risk of invisibility, scattering team to thin later, need to be more conscious of spreading the word

Tools

Bad tools discourage good practices
Buying a tool to make you agile
Buying testing tools
Using more tool than you need
Not automating the build on day 1

Copyright 2008 Atomic Object LLC

Bad tools discourage good practices

“We shape our tools and afterwards our tools shape us.”
Marshall McLuhan

Bend the tools to the practice, not vice versa

Source control

Languages

Suggestion: don't over-estimate the difficulty or under-estimate your developers

Copyright 2008 Atomic Object LLC

McLuhan: Canadian communication theorist, educator

Story: Visual Source Safe is setup so that each developer has their own repository. They don't commit even daily (no need), integration is infrequent, code lacks genuine source control.

Story: One big room, dev pairs, testing. SCC with locking means interruptions, manual hand offs, checking in non-compiling code.

Story: Language and unit test suite requires adding a new test to three places in two files. Developers make fat tests as a result. Code-generation helps solve.



Buying a tool to make you agile

It's easier to spend money than to think, learn, or change

Don't automate something you haven't done manually several times (at least)

Don't guess about what you'll need just to justify a tool

Copyright 2008 Atomic Object LLC

Tools don't make you agile
especially true for big, all-in-one, complicated,
religious-conversion tools



Buying testing tools

Not unique to, but a common agile adoption pitfall

Confusion goes like this:

agile means testing

testing is about finding bugs

regression testing requires automation

So we need to buy a testing tool!

Suggestion: build or borrow

Copyright 2008 Atomic Object LLC

I have heard the statistic widely quoted that 40% of
purchased testing tools sit on the shelf, unused

Testing in agile is a whole lot more than finding bugs

It's more of a development activity



Using more tool than you need

Two common errors

- delusions about what you're going to need
- feeling you need to build your own tool

Start with the simplest thing that could possibly work

Escalate only after you feel some pain

Have a range of tools

Exception: not automating the build first

Copyright 2008 Atomic Object LLC

like the simple design practice, don't assume you're gonna need it

index cards, whiteboards, paper, daily meeting

AO story tracking: index cards -> time tracking tool -> BaseCamp -> ExplainPMT



Project Status Tracking



Copyright 2008 Atomic Object LLC

from low-tech, easy to more complicated

cards and colored labels

time tracking tool

BaseCamp collaboration service

ExplainPMT web app



Not automating the build on day 1

Push button builds start paying off immediately

Manual builds make it hard to flex staff

Hard to find the time later

Project build knowledge is essential and should be explicit

Suggestion: favor build code over READMEs

Copyright 2008 Atomic Object LLC

payoff from first day - why not start immediately?

customer more understanding about ramp up than slow down

build code is better than a README (the one-line README)



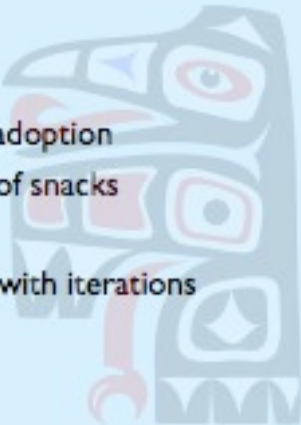
Magic Totems

Shallow iteration adoption

Missing the point of snacks

Holy index cards

Confusing phases with iterations



Copyright 2008 Atomic Object LLC

Shallow iteration adoption

Declaring an iteration period (1, 2, 3 weeks) but...

- not delivering software at the end
- not having customer prioritize development
- consistently accepting more work than can be done
- not completing the work you identify
- not testing the work you tackle

Don't expect to benefit from iterative development

Copyright 2008 Atomic Object LLC

probably not worse than before, but also not much better

Missing the point of snacks

It's not about free food

The point: communal activity, moving, sharing, talking, bonding, brainstorming

Suggestion: bulk snacks, a separate place

Copyright 2008 Atomic Object LLC

space doesn't have to be a lounge - just a different part of the room

Story: company learns snacks are "XP". buys individually wrapped snacks. developers take snack, eat alone at desk. Snacks first to be cut in budget woes.



Holy index cards

Adopting the distinctive elements of agile development doesn't bring you the benefits of agile practices

Cargo cults

Copyright 2008 Atomic Object LLC



Confusing phases with iterations

Phase: determined by business needs

Iteration: determined by development needs

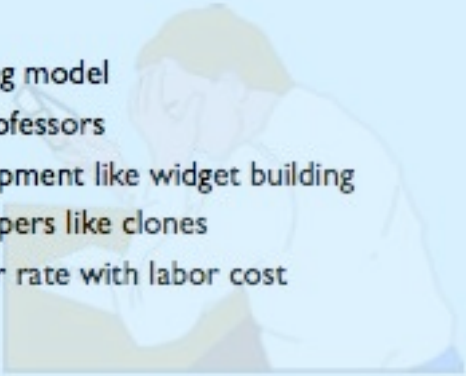
Suggestion: be strict in your consistent use of the terms

Copyright 2008 Atomic Object LLC

the concepts are distinct, both important

confusing the terms confuses the concepts

Misconceptions



- Using the wrong model
- Trusting the professors
- Treating development like widget building
- Treating developers like clones
- Confusing labor rate with labor cost
- Team size

Copyright 2008 Atomic Object LLC

Using the wrong model

Software Engineering comes from

- Engineering projects (hardware, software, systems)
- Huge scale (1000s of person years)
- A time of low-level languages and tools

What to do with idle programmers?

The page nobody read

Suggestion: consider software craftsmanship

Copyright 2008 Atomic Object LLC

the days of relatively low-level languages and tools

Pete McBreen's book on Software Craftsmanship is a good starting point

Winston Royce
“Managing the Development of Large Software Systems”
IEEE WESCON, August 1970

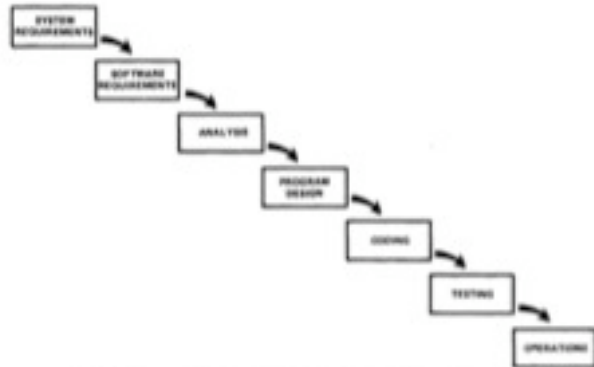


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Copyright 2006 Azamir, Object LLC



Page 2

"I believe in this concept, but the implementation described above is risky and invites failure."

Winston Royce
“Managing the Development of Large Software Systems”
IEEE WESCON, August 1970

Copyright 2006 Azamir, Object LLC





Trusting the professors

Common to see a nearly reflexive assumption that the way it's taught at university is an achievable, effective ideal (academic inadequacy)

The vicious cycle

- Trying to do X (some waterfallish practice)
- Failing on the project
- Castigating self for not being more disciplined
- Vowing to do more of X next time

Suggestion: understand and talk up the discipline of agile

Copyright 2006 Azamir, Object LLC

agile as “just hacking”, undisciplined, ad-hoc



Treating development like widget building

Building widgets or software both require

- engineering and design
- manufacturing

What this says about specialization of roles

Essay by Jack Reeves from The C++ Journal in 1982 sums up these ideas very nicely

Copyright 2006 Azamir, Object LLC

the problem: similar on the surface

the point: software is all “hard stuff” , manufacturing is trivial

design takes place while you're programming, whether you acknowledge it or not

See Bob Martin's Agile Software Development book for a copy



Treating developers like clones

Companies may treat developers as substitutable units according to the TLAs on their resumes, or the certification of their processes

Results in:

- seat-in-butt contracting
- forming and destroying internal teams per project
- focusing on hourly rate
- failed outsourcing

Copyright 2008 Atomic Object LLC

developers are people
the difference in most talented and average is dramatic

ultimately it all comes down to good people




Confusing labor rate with labor cost

Labor rate -- the cost of an hour of work

Labor cost -- the people portion of the cost of getting a system built

Copyright 2008 Atomic Object LLC



Team size

Assuming you need a large team

QSM study shows otherwise

Copyright 2008 Azamir Object LLC

Particularly egregious with a high-functioning agile team

QSM Study

Consultancy specializing in measuring, estimating, and controlling software development


- Database of 4000+ projects
- 2005 study on schedule vs team size
- 564 information systems projects since 2002
- Divided into small (< 5) and large (> 20) by team size

For projects of 100,000 SLOCs

- Average peak staffing of project: 32 (large), 4 (small)

Total effort for projects (person months)

- 178 for large teams (\$2.1 M)
- 25 for small teams (\$0.3 M)



Copyright 2008 Azamir Object LLC

Question: did the large teams finish faster?

Results

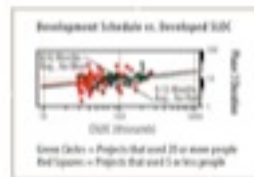
Calendar time to complete project

9.12 months for large team

8.92 months for small team

The one week shaved off delivery cost **\$1.8M**

Explanations?



Explanations?

Communication and coordination inefficiency

Greater rate of defects (5x)

Source:

“Haste makes waste when you over-staff to achieve schedule compressions”

Doug Putnam, QSM, Inc.

People

One-eyed kings in the land of the blind

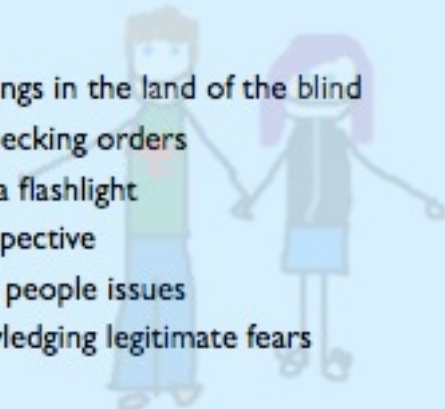
Disrupting pecking orders

Agile is like a flashlight

Lack of perspective

Ignoring the people issues

Not acknowledging legitimate fears





One-eyed kings in the land of the blind

Smart people viewed as successful, effective locally may claim to know agile practices

- when they've only read some books
- and won't try them ("done that for years...")
- may covertly work to oppose them (threat reaction)

Suggestion: recruit them

Copyright 2008 Atomic Object LLC

Often use names like "agilistas", refuse to adopt new terminology



Disrupting pecking orders

Customers like early-and-often

Managers like 10x fewer bugs

De-throning the one-eyed kings is problematic

- they don't take it lightly
- you usually still need their expertise
- you may need their capacity
- you may not be able to get rid of them

Copyright 2008 Atomic Object LLC

almost no matter who you are: consultant, junior developers, team lead, manager

you'll find problems with disturbing the pecking order

agile development practices are disruptive

Agile is like a flashlight

Shining a flashlight in dark corners reveals scary things

Cockroaches are survivors

Suggestion: be prepared to manage

Copyright 2008 Atomic Object LLC

pairing, one big room, build and test automation - people know what you're doing, what you know, what you're weak in

be prepared to handle personnel problems - agile won't do that

Lack of perspective

A team or even a whole company can have a very narrow perspective. Danger signs include

- low turnover
- limited exposure to new ideas
- belief that things are different for them
- pride and confidence in their track record, abilities

Suggestion: leadership required

Copyright 2008 Atomic Object LLC

Such a company may do pretty well, probably doesn't have major disasters

Not Invented Here - common reaction

Story: I've heard of companies that construct committees to "evaluate" new ideas, convince themselves that they couldn't benefit from change, spend a lot of time protecting the corporate ego

Risk: losing out on doing even better, doing it more efficiently, having more fun

leadership: challenging them to do better, travel budgets,



Ignoring the people issues

TDD, simple design, pair programming, refactoring, continuous build

The bad news

People, politics, management, communications, customers

Suggestion: hire for and train to the soft skills

Copyright 2006 Atomic Object LLC

The bad news is that the engineering practices are the easier part

The people, politics, management, communications, customers are the hard part

Agile exposes developers to customers. This means you need developers that have broader skills including the “softer” stuff



Not acknowledging legitimate fears

Agile returns the human element to software development

Developers and customers are human. Humans have legitimate fears

Not acknowledging and addressing these fears is risky

Copyright 2006 Atomic Object LLC

this is why we like the craftsmanship model

Source: “Planning Extreme Programming” by Kent Beck, Martin Fowler

Customers fear...

will ask for the wrong things

won't get what they asked for

will pay too much for what they get

won't know where the project really stands

won't be able to change their minds if their business changes

Copyright 2008 Atomic Object LLC



Developers fear...

being asked to do more than they can in a given time period

being asked to do things they don't know how to do

being asked to solve hard problems alone

being asked to do things they know are wrong

being asked to do things they know are a silly waste of time

being given responsibility but no authority

Copyright 2008 Atomic Object LLC

