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
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

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?"
Asking them for examples, help them sketch things, describe scenarios.

whiteboards, paper, document

Question: Do you keep these artifacts?
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

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?

Letting the stronger person drive

Two roles: driver, navigator

A strong person driving must be careful

Suggestion: weaker partner drives, or switch frequently
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")

Only pairing when introducing new team members

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

---

**TDD and Deadlines**

---

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

---

**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

---

**Characteristics of good tests**

- A - automated
- T - thorough
- R - repeatable
- I - independent
- P - professional
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?)

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?
Using “implement me” for more than a day

The test you recognize needs to be written

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

People stop expecting the green bar

---

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

---

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

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

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

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
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

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

**Agile developers need company**

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

Suggestion: pairs are powerful

---

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

**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
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
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
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

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

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
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

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

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

the ultimate job of leadership
Managers were good at doing what the legacy system valued.
Changing that system causes legitimate fears.
Capital One: Agile Int. Conf, 2006

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

Making bogeymen of external forces

“But the auditors said...”
SAS70, HIPPA, CMM, EVM, DO-178B

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

Customer is part of the team
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
Scrum has definitely won the marketing game (vs XP’s planning game)
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%)

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

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

\[ V_{\text{new}} = \alpha \times V_{\text{latest}} + (1-\alpha) \times V_{\text{old}} \]
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

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
### Design

Not having good enough design skills  
Have a pattern, find a need  
The singleton pattern

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Lines</th>
<th>LOC</th>
<th>Classes</th>
<th>Methods</th>
<th>M/C</th>
<th>LOC/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helpers</td>
<td>171</td>
<td>149</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>Controllers</td>
<td>660</td>
<td>680</td>
<td>16</td>
<td>95</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>RTIs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Components</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Functionals</td>
<td>1750</td>
<td>1344</td>
<td>35</td>
<td>127</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Models</td>
<td>393</td>
<td>326</td>
<td>22</td>
<td>41</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Units</td>
<td>743</td>
<td>559</td>
<td>17</td>
<td>58</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

Total: 3617 | 3064 | 96 | 315 | 3 | 7 |

Code LOC: 1161  Test LOC: 1903  Code to Test Ratio: 111.6

---

Finished in 62.6977 seconds.
80 tests, 687 assertions, 0 failures, 0 errors

Copyright 2006 Atium Object LLC
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

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
dependency injection framework can help with composition of objects

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

Business Model

Development time is free
Asking customers to prioritize cost, quality, scope
Fixed price agile development
Really small projects
ironically this is an advantage of using an outside contractor

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

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 company, cost of legacy
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

Agile conferences the last few years: scaling agile up
My interest is the opposite: scaling agile down
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?

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"

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

risk of incremental change
nothing much changes
new problems aren’t addressed
skeptics see it as a passing fad
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
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.
Tools don’t make you agile especially true for big, all-in-one, complicated, religious-conversion tools

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
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

from low-tech, easy to more complicated cards and colored labels
time tracking tool
BaseCamp collaboration service
ExplainPMT web app
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)
probably not worse than before, but also not much better

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.
the concepts are distinct, both important
confusing the terms confuses the concepts

Holy index cards

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

Confusing phases with iterations

Phase: determined by business needs
Iteration: determined by development needs
Suggestion: be strict in your consistent use of the terms
the days of relatively low-level languages and tools

Pete McBreen’s book on Software Craftsmanship is a good starting point
"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
agile as “just hacking”, undisciplined, ad-hoc

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
developers are people
the difference in most talented and average is dramatic
ultimately it all comes down to good people

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

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
Particularly egregious with a high-functioning agile team

Question: did the large teams finish faster?

Team size

Assuming you need a large team

QSM study shows otherwise

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)
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.
Often use names like “agilistas”, refuse to adopt new terminology

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 covertsly work to oppose them (threat reaction)

Suggestion: recruit them

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

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
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

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,
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

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

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