14 Years of TDD
What We've Learned
at Atomic Object

Carl Erickson
@carl_erickson
GreatNotBig.com
GREAT NOT BIG
ON BUILDING AND RUNNING A SOFTWARE DEVELOPMENT COMPANY

Blog  Speaking  About  Contact

Adventures in ownership

BY CARL ERICKSON | DECEMBER 26, 2012

It was about a year ago that I described employee ownership of a company as a "partial emergent order". An emergent order is a system that arises between the interactions of many independent components with no central control. Markets are emergent orders. Made orders are systems created with rules and central control. Companies are made orders. I have an abiding interest in this distinction because I've been heading Atomic down a path of significant employee ownership for the last four years.

2012 was a big year for Atomic Object on the ownership front:

• we declined an opportunity to sell the company to outsiders
• we broadened our employee ownership from 8 to 15 people
• we said goodbye to an owner for the first time
• I went from 63% ownership to 53% ownership
• Shawn and Mike laid the groundwork to launch an Employee Stock

To in-source or outsource? That is the question

How should you build your next innovative product or service? One major consideration is whether to do the work inside your company or outsource it. I've identified some key dimensions of this problem to help you think through your choice. I'm assuming you have a project large enough to need at least a small team of people, that the stakes are high for you and your company, that time-to-market matters, but is not the overriding factor and that your company is large enough to have employees to consider using.

What to consider

Technical expertise

Does your team have the requisite technical skills? If not, can they effectively learn them quickly enough? Don't forget that there are two aspects to this dimension: expertise in the technology of

@carl_erickson
web

@carl_erickson
Support the tasks and foster social engagement.
<table>
<thead>
<tr>
<th>TECHNOLOGY STARTUPS</th>
<th>MANUFACTURING</th>
<th>EDUCATION, GOVERNMENT &amp; NON-PROFIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>sidereel</td>
<td>Whirlpool®</td>
<td>Willow Creek Association</td>
</tr>
<tr>
<td>Bloomfire</td>
<td>x-rite</td>
<td>CITY OF GRAND RAPIDS</td>
</tr>
<tr>
<td>realms</td>
<td>Herman Miller</td>
<td>ArtPrize</td>
</tr>
<tr>
<td>Kidtelligent</td>
<td>Eaton</td>
<td>National Heritage Academies</td>
</tr>
<tr>
<td>reapso</td>
<td>Chrysler</td>
<td>ACTON Institute</td>
</tr>
<tr>
<td>bluemedora</td>
<td>Gentex</td>
<td>Grand Rapids Medical Education</td>
</tr>
<tr>
<td>Medical Micrographics</td>
<td></td>
<td>Partners</td>
</tr>
<tr>
<td></td>
<td>Stryker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Steelcase</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Faurecia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>John Deere</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dematic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority Health</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spectrum Health</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Advanced Radiology Services</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Covenant</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retirement Communities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMERICAN SEATING</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thursday, February 27, 14
Atomic Value Mantras

- Give a shit
- Share the pain
- Teach and learn
- Own it
- Act transparently
Our Testing Journey

- We’ve been at this for 14 years
- We’ve made plenty of mistakes
- We’ve learned a few things
Design validation
Usability testing
Exploratory tester
Behavior driven development
Testing DB migrations
Pure TDD for unit tests
Test-parallel unit tests
By devs, automated, chunk at a time
By devs, automated, when “done”
A separate QA team
One-off main()s
#1 Everyone hits the wall

- The bigger your brain, the taller the wall
- You need tests, past the wall
#2 Learn what to test

- Formal name, simple idea: equivalence classes
- It’ll help you understand what you should test
#3 Your unit tests will grow fat

- Don’t give up when your state-based, bloated unit tests become too expensive to maintain
- Learn about stubs
- Learn about interaction testing and mocks
- Learn about dependency injection
#4 Know when to stop

- There’s always another test you can write
- Testing is an interesting, tricky problem; don’t fall blindly in love with it
- Testing is an economic activity
- Consider risk, cost and context
#5 Design for testability

- Some software designs are much harder to test than others; design for testability
- Automating your test pushes you toward learning and understanding good design principles and patterns
#6 Test-infected developers don’t need to be “managed”

- TDD makes everybody’s life better
  - developer, tester, manager, customer
#7 System tests are harder

- Don’t try to cover all the cases with system tests
- GUIs, external connections, operating systems, web browsers - hard
- Design for testability is critical here
#8 Toolsmithing helps

• The techniques of code generation, helper functions, and general toolsmithing can help with bulky testing situations

• Some languages make this easier than others (Ruby vs C, for instance)
#9 Testing legacy code is no fun

- It almost certainly wasn’t designed for testability
- Create system, not unit, tests
- Isolate, replace or refactor problematic sections with TDD
- And of course test-drive all new code
#10 Test code is real code

- It needs to be robust and reliable
- It needs to be maintained (regular refactoring)
- Invest in your tests like you invest in your source
Files will multiply

- Don’t let that stop you
- Improve your tools
#12 Dev testing is not enough

- 10x improvement is seductive
- Exploratory testing is a human-worthy activity
- Usability, inconsistencies, browser variance, integration issues, and pretty rarely, plain old bugs
#13 Design validation eliminates bugs

- The code you don’t write has no bugs
- This form of “testing” is about quality, more broadly defined
- Software that delights users, is easy to use, isn’t prone to user errors, and address user needs is a huge quality win