

# eXtreme Programming, Open Source, and Geographically Dispersed teams

Michael Richardson  
<[mcr@xelerance.com](mailto:mcr@xelerance.com)>

Xelerance Corporation

<http://www.xelerance.com/>

me:

<http://www.sandelman.ca/mcr/>

talk: <http://www.xelerance.com/talks/bsdcan2004/>



# Overview of talk

- Introduction to speaker
- Introduction to eXtreme Programming
- The victim project
- Challenges of XP and Open Source
- Challenges of XP and telecommuting
- Tools and experiences
- Conclusions



# Introduction to speaker

- Michael Richardson <mcr@xelerance.com>
- Long time Unix, BSD, Linux user.
- I write network protocol stuff: IPsec, radius, telnet, ssh, etc.
- history of security stuff: milkyway.com (firewalls), solidum.com (L2-L7 policy-based classification), SSH (IPsec-Express), other stuff.
- VP R&D at Xelerance Corporation
- Xelerance.com is providing 3rd level defect support for Openswan.
- AKA Sandelman Software Works, also tcpdump.org maintainer.
- This talk at <http://www.xelerance.com/talks/bsdcan2004/>



# Introduction to eXtreme Programming

- no revolution here – just evolution
  - turns out I've been doing 70% of it all along
  - I'll bet you do that much already
  - nice manager friendly names for things we already knew
- some books by X
- websites: <http://www.extremeprogramming.org/>



# 10 key elements of eXtreme Programming

- user stories
- iterations
- unit tests
- refactoring
- pair programming
- acceptance tests
- daily stand up meeting
- integrate often
- release often
- fix XP when it breaks



# User stories

- the user/customer defines what is important.
- they define this via a “story”, or scenario of some kind.
- e.g. “I want to be able to type  
`pkg_info -vuln-check apache2`”



# Iterative Development

- features are added one at a time
- user/customer is allowed to provide feedback
- entire system is always useable in some fashion at end of iteration
- short iterations. 2 weeks to 2 months.



# Unit tests

- all code has a unit test
- unit test confirms that one function/program/subsystem works
- provides for typical inputs/outputs
- is easy to run
- used sanity check
- added when bugs are found





# Refactoring

- code is rewritten/broken up as needed
- test cases confirm that code is correct
- 



# Pair Programming

- two brains, one keyboard/screen
- allows for strategic/tactical thinking
- spreads out the knowledge
- makes new experts
- natural mentoring
- teaches old dogs new tricks



# Acceptance Tests

- a test case which the user/customer uses to determine if feature is correct
- the acceptance test tells the programmer if they are finished yet
- explains the requirements in a brief fashion



# Daily stand up meetings

- communicates current status quickly
- replaces long status meetings
- provides for quick response when people are stuck
- 



# Integrate Often

- merge all code into HEAD regularly
- refactored code has unit tests – they pass first.
- systems have acceptance tests



# Release Often

- the customer never has to wait long
- fast response to customer need
- no custom/one-off releases for one-customer
- increased enthusiasm, feeling of success



# fix XP when it breaks

- none of this is written in stone
- fix the process when it is wrong
- evolution – not revolution.



# Victim Project

- Linux FreeS/WAN IPsec project
- hard to test stuff
- initially 3 in Ottawa, 3 in Toronto, 1 “manager” on Earth, 1 “funder” in San Francisco
  - US export/influence situation
- lots of security for privacy (STU-III “batphones” initially)
- many different networks
  - mine is 70% NetBSD, 30% Linux. All critical boxes are NetBSD. Similar mixes elsewhere.





# Why talk about this here?

- Linux experience at BSDcan?
- FreeS/WAN project was closer to \*BSD than Linux kernel in philosophy
  - project leads trying to bring BSD quality to Linux networking
  - structure of \*BSD is simpler than Linux – one project, not hundreds of distros, vs kernel.
- (Net)BSD project(s) have frequent issues/debates with quality/scope/schedule.



# Testing infrastructure

- I spent 2001/2002 was spent working on testing infrastructure. (In advance of switch to XP)
- host based simulation via User-Mode-Linux rather than 5 box system
- tests are repeatable, self-contained and portable
- strong preference for “dog-food” - we transitioned in 2001/2002 to running our code everywhere – including CVS access.



# Transition to XP

- started monthly telephone conferences via multi-line POTS+batphone
- switched to H.323 (“GnomeMeeting”) in Winter 2002. Over IPsec!
- switched to SIP in June 2003. Over IPsec.
- team-only IRC channel (Over IPsec)
- public IRC channel on [irc.openprojects.net](http://irc.openprojects.net) -> [irc.freenode.net](http://irc.freenode.net).



# Transition to XP (2)

- bought books in March 2002
  - some confusion as to whether we were “doing XP” yet. Transition to XP can be hard.
  - full blown by August 2002.
- all current tasks were broken up into stories.
- Developers were asked to provide estimates of effort required.
- “customer” was asked to rank things
- we wrote an iteration schedule, and stuck to it.



# Velocity Measure

- velocity is a measure of “useful work time” to “real time”
- takes in account interruptions, email, being sick, re-installing Windows (!), etc.
- this is measured empirically



# Challenges of XP and Open Source

- who is the customer?
- need them for acceptance tests/sign-off
- need them for ranking of work
- who is the responsible for testing?
- who is the “project manager”? (do we need one?) (do they have enough authority?)



## Who is the customer

- we are the customers.
  - compiler people are customers of kernel people
  - kernel people are customers of compiler people
  - all of them work under X-windows
  - libc people are customers of kernel
- you make up the relations



## Who is the customer

- actually points to a lack – we have developers and we organize them!
- but we don't have organizations of customers!
- Linux has this – Distro companies provide for the customers. Where are the equivalents for \*BSD?
- there are some, but not well known.





## Who is the customer

- FreeS/WAN project had political problems – the guy with the money felt he was not allowed to provide technical input!
- had to improvise a bit.



# Challenges of XP and telecommuting

- stand-up meetings
- pair programming
- integrate often
- iterations
  - vastly different velocity measures



# Stand-up Meetings

- despite being in the same timezone, we were almost always in different phases.
  - worse when real time zones are involved as well
  - also worse when developers are not paid!
- agreed to “be around” at 13:30 every day.
  - At least in IRC.



# Stand Up Meetings

- VoIP when we need it.
- iteration meeting is well announced
  - often 3-4 hours long
  - once every three weeks (our iteration cycle)
  - near end of current iteration
- 45 minutes allocated to “technical” issues with meeting.
  - eat your own dog food
  - many rekey and DNS related bugs in IPsec found by using it ourselves



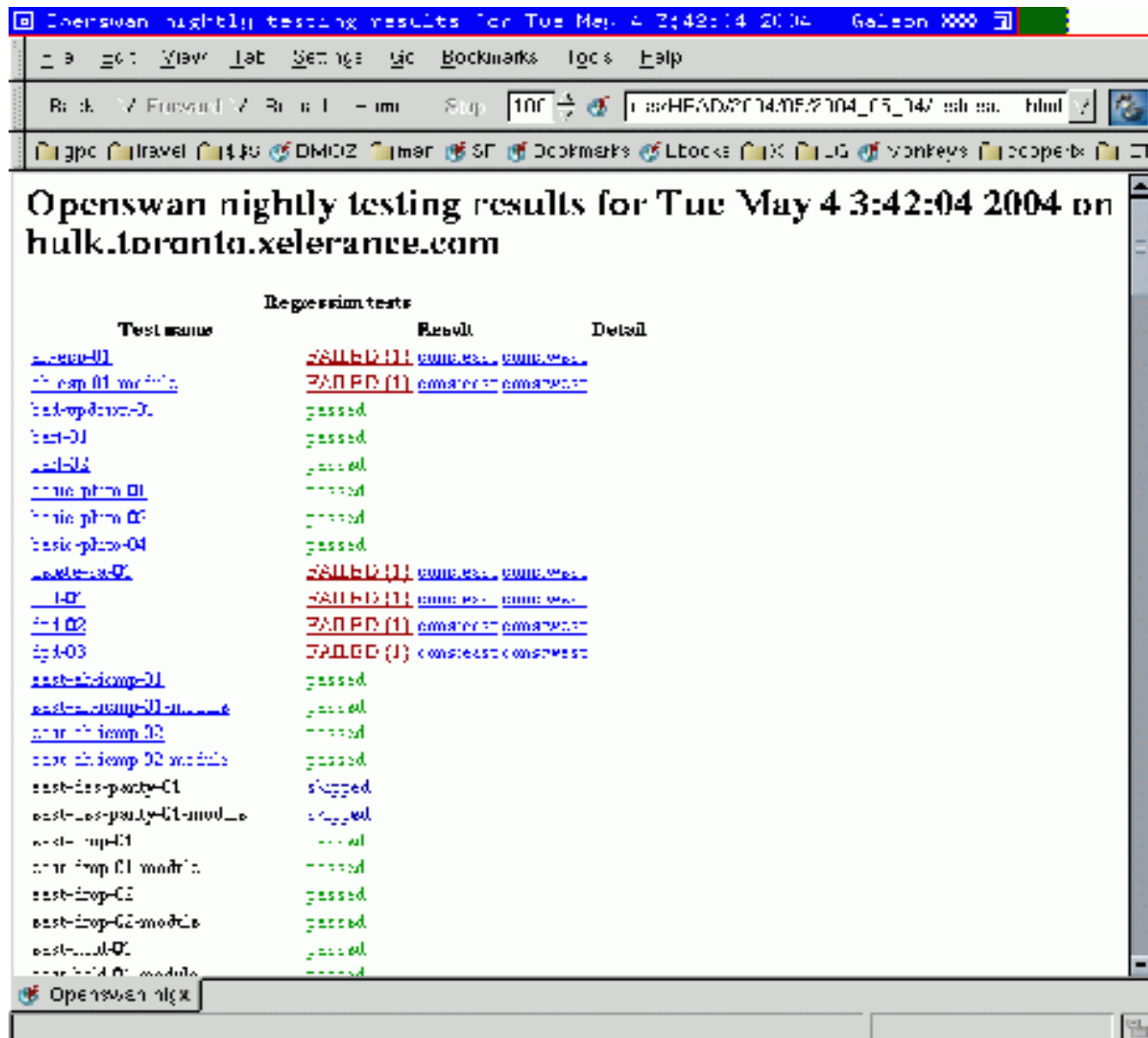
## testing – when to complain

- testing system issues daily email to all
- lists tests that succeed, those that fail, and permits deeper analysis
- often culprit would know they were responsible.
- but, sometimes didn't have time to fix.
- DERIVED RULE:

**all test cases pass at end of iteration**



# testing – example output



Openswan nightly testing results for Tue May 4 3:42:04 2004 on hulk.toronto.xelerance.com

Test name	Result	Detail
<a href="#">basic-01</a>	FAILED (1)	<a href="#">conn.exe</a> <a href="#">conn.exe</a>
<a href="#">basic-01-module</a>	FAILED (1)	<a href="#">conn.exe</a> <a href="#">conn.exe</a>
<a href="#">basic-02</a>	passed	
<a href="#">basic-03</a>	passed	
<a href="#">basic-04</a>	passed	
<a href="#">basic-05</a>	passed	
<a href="#">basic-06</a>	passed	
<a href="#">basic-07</a>	passed	
<a href="#">basic-08</a>	passed	
<a href="#">basic-09</a>	passed	
<a href="#">basic-10</a>	FAILED (1)	<a href="#">conn.exe</a> <a href="#">conn.exe</a>
<a href="#">basic-11</a>	FAILED (1)	<a href="#">conn.exe</a> <a href="#">conn.exe</a>
<a href="#">basic-12</a>	FAILED (1)	<a href="#">conn.exe</a> <a href="#">conn.exe</a>
<a href="#">basic-13</a>	FAILED (1)	consistent consistent
<a href="#">basic-14</a>	passed	
<a href="#">basic-15</a>	passed	
<a href="#">basic-16</a>	passed	
<a href="#">basic-17</a>	passed	
<a href="#">basic-18</a>	passed	
<a href="#">basic-19</a>	passed	
<a href="#">basic-20</a>	passed	
<a href="#">basic-21</a>	passed	
<a href="#">basic-22</a>	passed	
<a href="#">basic-23</a>	passed	
<a href="#">basic-24</a>	passed	
<a href="#">basic-25</a>	passed	
<a href="#">basic-26</a>	passed	
<a href="#">basic-27</a>	passed	
<a href="#">basic-28</a>	passed	
<a href="#">basic-29</a>	passed	
<a href="#">basic-30</a>	passed	
<a href="#">basic-31</a>	passed	
<a href="#">basic-32</a>	passed	
<a href="#">basic-33</a>	passed	
<a href="#">basic-34</a>	passed	
<a href="#">basic-35</a>	passed	
<a href="#">basic-36</a>	passed	
<a href="#">basic-37</a>	passed	
<a href="#">basic-38</a>	passed	
<a href="#">basic-39</a>	passed	
<a href="#">basic-40</a>	passed	
<a href="#">basic-41</a>	passed	
<a href="#">basic-42</a>	passed	
<a href="#">basic-43</a>	passed	
<a href="#">basic-44</a>	passed	
<a href="#">basic-45</a>	passed	
<a href="#">basic-46</a>	passed	
<a href="#">basic-47</a>	passed	
<a href="#">basic-48</a>	passed	
<a href="#">basic-49</a>	passed	
<a href="#">basic-50</a>	passed	
<a href="#">basic-51</a>	passed	
<a href="#">basic-52</a>	passed	
<a href="#">basic-53</a>	passed	
<a href="#">basic-54</a>	passed	
<a href="#">basic-55</a>	passed	
<a href="#">basic-56</a>	passed	
<a href="#">basic-57</a>	passed	
<a href="#">basic-58</a>	passed	
<a href="#">basic-59</a>	passed	
<a href="#">basic-60</a>	passed	
<a href="#">basic-61</a>	passed	
<a href="#">basic-62</a>	passed	
<a href="#">basic-63</a>	passed	
<a href="#">basic-64</a>	passed	
<a href="#">basic-65</a>	passed	
<a href="#">basic-66</a>	passed	
<a href="#">basic-67</a>	passed	
<a href="#">basic-68</a>	passed	
<a href="#">basic-69</a>	passed	
<a href="#">basic-70</a>	passed	
<a href="#">basic-71</a>	passed	
<a href="#">basic-72</a>	passed	
<a href="#">basic-73</a>	passed	
<a href="#">basic-74</a>	passed	
<a href="#">basic-75</a>	passed	
<a href="#">basic-76</a>	passed	
<a href="#">basic-77</a>	passed	
<a href="#">basic-78</a>	passed	
<a href="#">basic-79</a>	passed	
<a href="#">basic-80</a>	passed	
<a href="#">basic-81</a>	passed	
<a href="#">basic-82</a>	passed	
<a href="#">basic-83</a>	passed	
<a href="#">basic-84</a>	passed	
<a href="#">basic-85</a>	passed	
<a href="#">basic-86</a>	passed	
<a href="#">basic-87</a>	passed	
<a href="#">basic-88</a>	passed	
<a href="#">basic-89</a>	passed	
<a href="#">basic-90</a>	passed	
<a href="#">basic-91</a>	passed	
<a href="#">basic-92</a>	passed	
<a href="#">basic-93</a>	passed	
<a href="#">basic-94</a>	passed	
<a href="#">basic-95</a>	passed	
<a href="#">basic-96</a>	passed	
<a href="#">basic-97</a>	passed	
<a href="#">basic-98</a>	passed	
<a href="#">basic-99</a>	passed	
<a href="#">basic-100</a>	passed	



# how to do pair programming?

- screen!
- has multiuser mode
- two people SSH to same box, start screen.
- get on VoIP or POTS to talk
- get on IRC in case VoIP fails



# Pair Programming (2)

- origin of Pair Programming from SmallTalk
- no compile cycle
- what to do while it compiles?
- answer: being in separate places with two keyboards, two screens, means one can multitask.
- not clear if this is a good thing!





# testing – getting everyone on the same page

- acceptance tests.
- we integrated too often!
- branches are hard. Maybe SubVersion will help.
- 



# release often

- we got better at this
- but we experienced “2.0 syndrome”, despite trying to avoid this.
- we did get better after .0



# Conclusions

- eXtreme Programming can work
- open source projects can benefit from these methods
  - velocity is a problem
  - paid/unpaid is a problem
- geography is a challenge, but also an opportunity



# Questions?

- me:
- <http://www.sandelman.ca/mcr/>
- talk:  
<http://www.xelerance.com/talks/bsdcan2004/>

