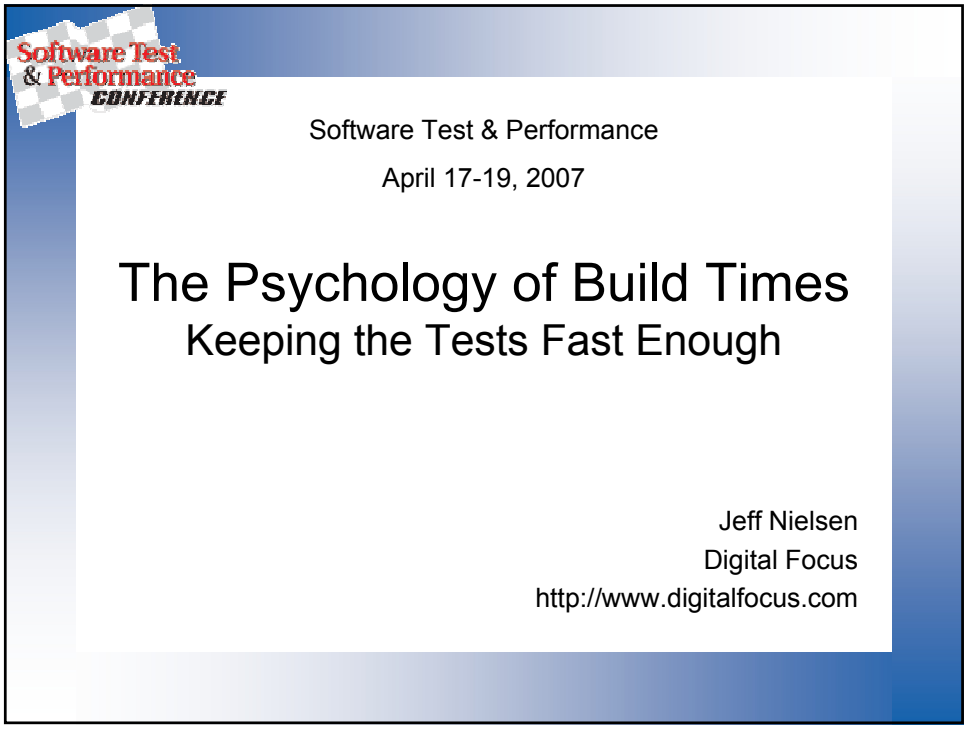
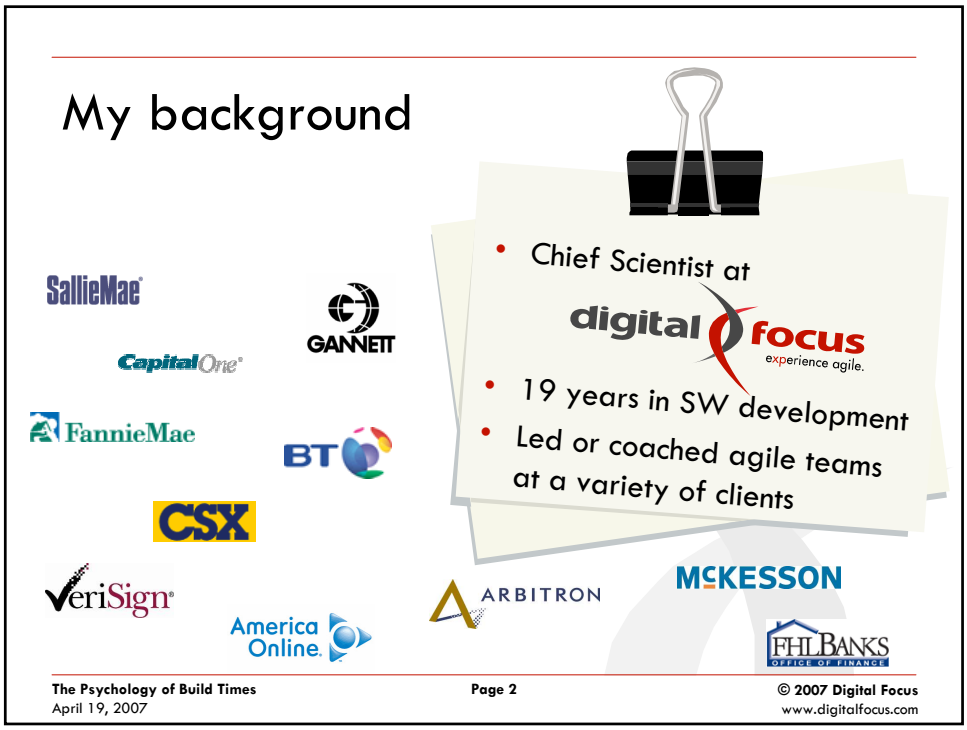
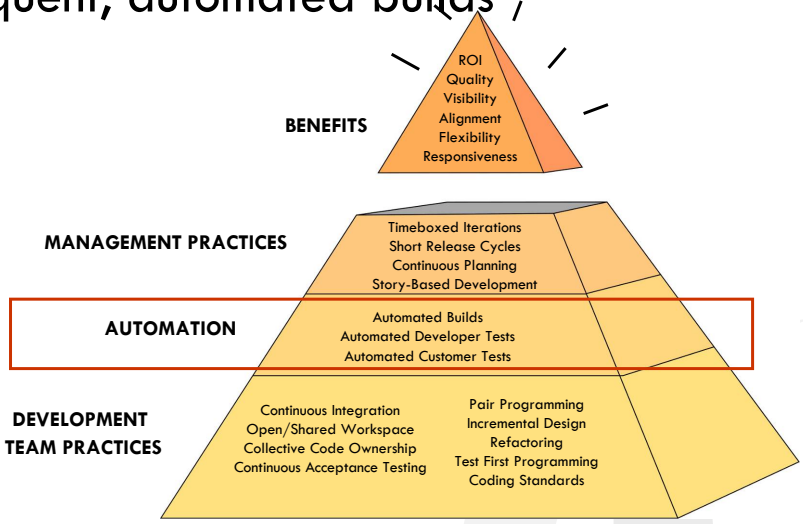
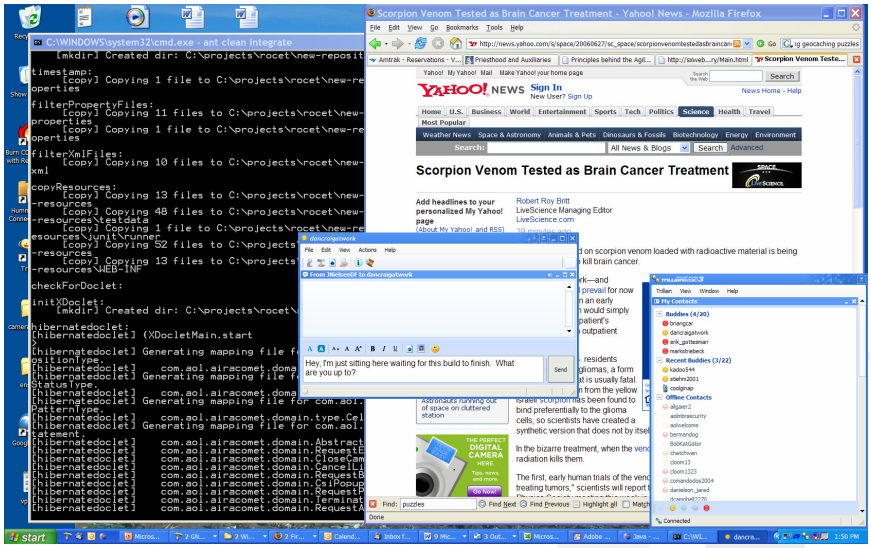

Software Test
& Performance
CONFERENCE
Software Test & Performance
April 17-19, 2007
The Psychology of Build Times
Keeping the Tests Fast Enough
Jeff Nielsen
Digital Focus
http://www.digitalfocus.com


My background
SallieMae
GANNETT
CapitalOne
FannieMae
BT
CSX
VeriSign
America Online
ARBITRON
McKESSON
FHLBANKS
OFFICE OF FINANCE
Chief Scientist at
digital focus
experience agile.
19 years in SW development
Led or coached agile teams at a variety of clients
The Psychology of Build Times
April 19, 2007
Page 2
© 2007 Digital Focus
www.digitalfocus.com

Agile methodologies rely heavily on frequent, automated builds



Build times affect the team's behavior



Exercise: Observe yourself

- Build 1: 20 seconds with no visible feedback
- Build 2: 45 seconds with feedback
- Build 3: 20 seconds with feedback
- Build 4: 12 seconds

Human psychology imposes natural limits

Based on your observation and personal experience . . .

- How much time is a developer willing to spend *running tests locally* after changing a few lines of code? — 5 – 10 sec.
- How much time is a developer willing to spend running tests locally when they are *ready to check in*? — 15 – 20 sec.
- How long is a developer willing to *sit and watch* the integration machine console *when integrating*? — 30 – 40 sec.
- How long is a developer waiting to hang around the team room/building *after checking in*? — 10 min.
- How much time will a tester wait (without starting something else) for a new build with a minor change they need? — 1 – 2 min.

You need different builds, optimized for different situations

Developer Build

EXAMPLE

Compile modified code
Execute "fast" unit tests

Developer Build

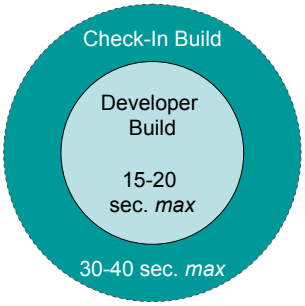
15-20 sec. *max*

Check-In Build

EXAMPLE

Compile modified code
Execute "fast" unit tests

Update from source control
Do clean compilation



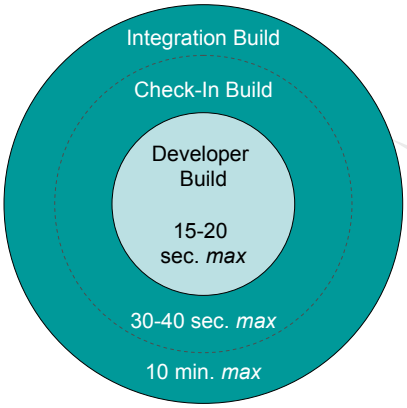
Integration Build

EXAMPLE

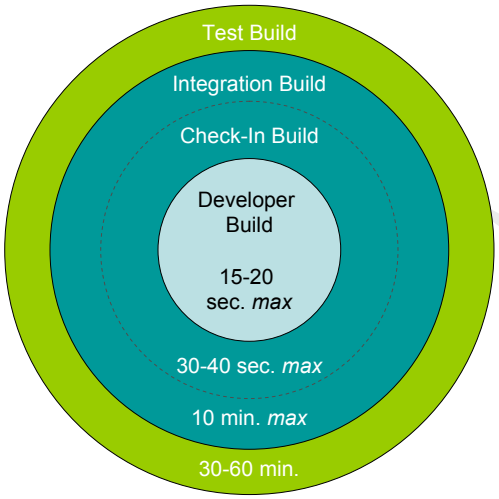
Compile modified code
Execute "fast" unit tests

Update from source control
Do clean compilation

Build application (e.g., EAR file)
Deploy/launch application
Run "slower" unit tests
Run non-GUI functional tests



Test Build



EXAMPLE

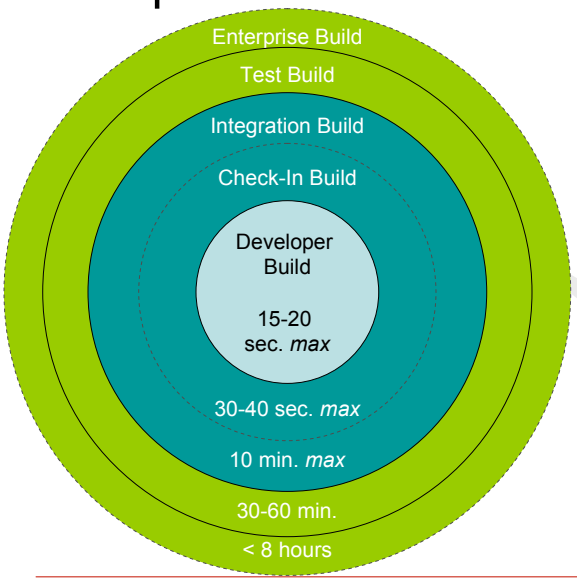
Compile modified code
Execute "fast" unit tests

Update from source control
Do clean compilation

Build application (e.g., EAR file)
Deploy/launch application
Run "slower" unit tests
Run non-GUI functional tests

Do clean build of application
Run GUI-based functional tests

Enterprise Build



EXAMPLE

Compile modified code
Execute "fast" unit tests

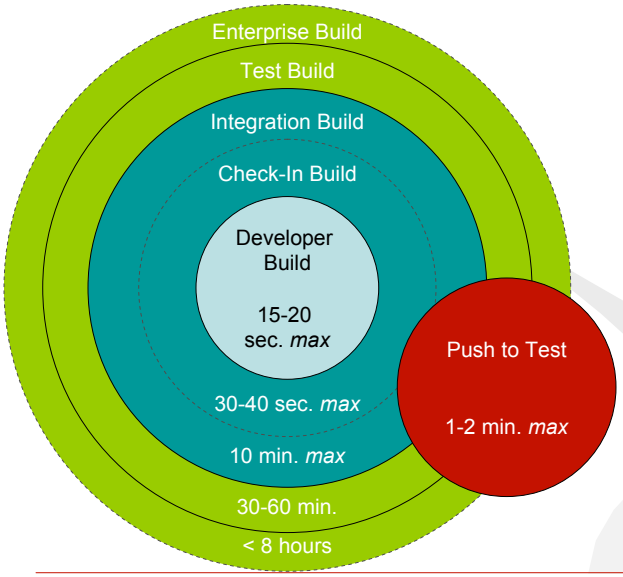
Update from source control
Do clean compilation

Build application (e.g., EAR file)
Deploy/launch application
Run "slower" unit tests
Run non-GUI functional tests

Do clean build of application
Run GUI-based functional tests

Integrate with systems from other teams

Push to Test



EXAMPLE

Compile modified code
Execute "fast" unit tests

Update from source control
Do clean compilation

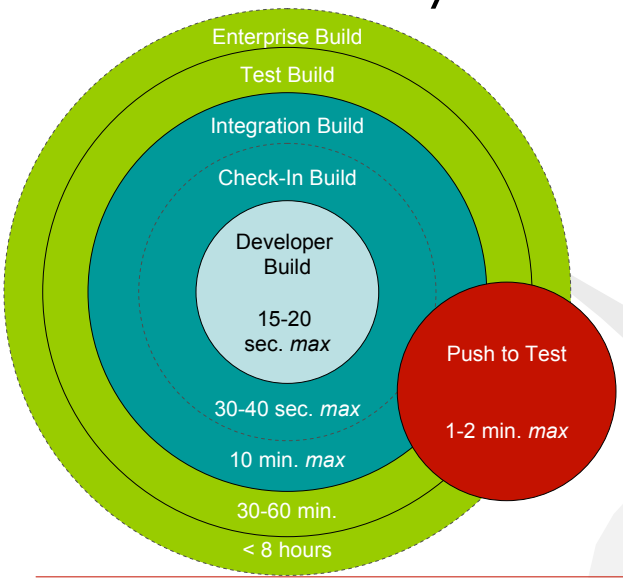
Build application (e.g., EAR file)
Deploy/launch application
Run "slower" unit tests
Run non-GUI functional tests

Do clean build of application
Run GUI-based functional tests

Integrate with systems from other teams

Move last build to test server
Re-deploy application

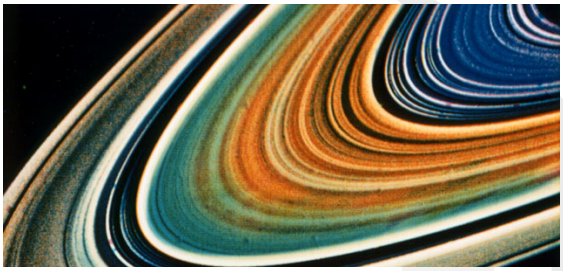
The "Build Solar System"



- Developer Build
  - Performed "on demand" on local machine
- Check-In Build
  - Performed "on demand" at each checkin
- Integration Build
  - Kicked off automatically on each checkin
- Test Build
  - Performs additional tasks to ensure testable system
- Enterprise Build
  - For a large system with multiple teams
- Push to test
  - Kicked off on demand or at pre-scheduled times
  - Draws from last successful test or integration build

Designing a build system

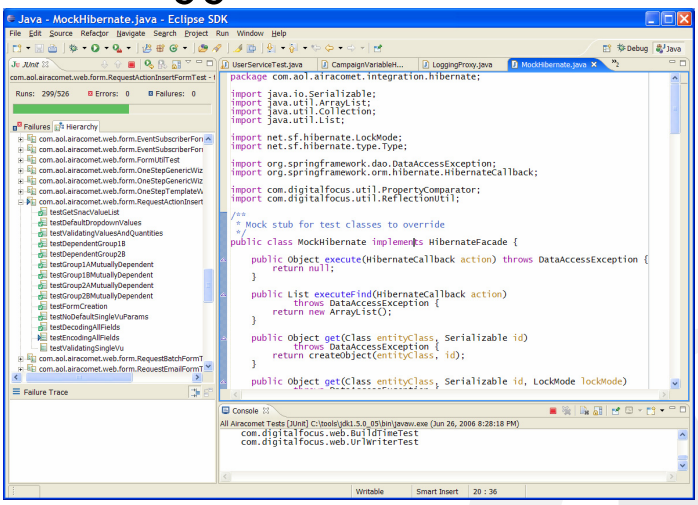
- Maximize the amount of work performed in each build "orbit" within the acceptable duration.
- Iteratively add build orbits as needed (refactoring)
- When a build orbit exceeds the threshold time, first try to optimize it
- *Not all orbits are needed for all projects.*



What slows down a build?

- Cumbersome processes
- Inappropriate technologies
- . . .

Poor management of automated tests is often the biggest bottleneck



Guidelines for fast unit tests

Michael Feathers,
"Unit Test Rulz"
http://xunitpatterns.com/Unit%20Test%20Rulz.html

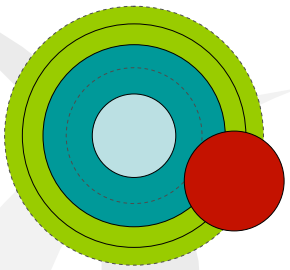
A test is not a (fast) unit test if:

1. It talks to the database
2. It communicates across the network
3. It touches the file system
4. It can't run correctly at the same time as any of your other unit tests
5. You have to do special things to your environment (such as editing config files) to run it

> Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can **keep a set of tests that we can run fast whenever we make our changes.**

A real-life build solar system

- Custom J2EE project for Fortune 500 client
- System integrates with legacy enterprise infrastructure
  - 10 other apps
- Weblogic with Oracle back end
- Hibernate & Spring frameworks
- Lots of Javascript & Ajax
- ~1300 Java source files



Tests grouped by naming convention

- xxxTest.java (1854)
  - fast unit tests
- xxxTestDb.java (610)
  - tests that talk to a database
- xxxTestCp.java (43)
  - other integration tests
- xxxTestJms.java (10)
- xxxTestSrv.java (2)
  - in-container Cactus tests

Airacomet Build Solar System

Orbit	Invoked with	Does	Avg. time
Developer Build	"ant test" or run configuration in IDE	• Re-compiles changed Java files • Re-filters changed config files • Runs "fast" unit tests	7-8 secs.
Check-in Build	"ant integrate"	• Fetches updates from CVS • Cleans build directory • Compiles Java files • Filters config files • Produces generate code/XML • Runs "fast" unit tests	35-40 secs.
Integration Build	"ant cruiseControlTests" (automatically upon checkin)	• Re-builds test database • Populates test database • Pre-compiles JSPs • Cleans build directory • Builds whole app (EAR file) • Generates clean WLS domain • Starts WebLogic server (new shell) • Runs "fast" unit tests • Runs DB, Cactus, and integration tests	7-8 mins.

Airacomet Build Solar System

Orbit	Invoked with	Does	Avg. time
Push to Test	"newbuild"	• Starts with latest check-in build on integration machine • Creates EAR file • FTPs EAR file to test server • Re-deploys EAR on test server	1 min.
Test Build	"ant runSeleniumTests"	• Runs integration build • Runs Selenium tests	45 mins.
Enterprise Build	(doesn't exist)		

Summary of principles

- Pay attention to your build design, the same way you do your system design
- Maximize the amount of *feedback* you can get from each build
  - Feedback is more valuable the sooner it arrives
  - Recognize tradeoff between thoroughness & speed
- When you have a choice, prefer faster tests over slower ones

Build times matter

Agile practices are most easily maintained

with optimized build times