On Software Release Engineering
Bram Adams
On average we deploy new code **fifty** times a day.
Continuous Delivery
Continuous Delivery

http://goo.gl/qPT6
Continuous Delivery

continuous integration

CVS
Continuous Delivery

continuous integration

test

http://goo.gl/qPT6
Continuous Delivery

continuous integration

9 min.

15k tests
Continuous Delivery

CVS

test

15k tests

staging/production

continuous integration

9 min.
Continuous Delivery

CVS

continuous integration

9 min.

15k tests

staging/production

http://goo.gl/qPT6
Continuous Delivery

Continuous integration

CVS

15k tests

test

staging/production

9 min.

6 min.

http://goo.gl/qPT6
Work fast and don’t be afraid to break things.
Even Desktop Apps Release More Frequently

... yet Software Systems keep on Growing!

... yet Software Systems keep on Growing!

>5k developers
>2k projects
>50M tests/day
>50k builds/day

<150k tests/commit

>50TB memory

20 code changes/min.

build cache with

compilation in the cloud

Release Engineering
Release Engineering

in-house/3rd party development
Release Engineering

in-house/3rd party development

integration
Release Engineering

in-house/3rd party development

integration

deployment
Release Engineering

in-house/3rd party development

integration

reduce cycle time!

deployment
Release Engineering

in-house/3rd party development

integrate

build

test

reduce cycle time!

deployment
Release Engineering

in-house/3rd party development

integrate

build

test

reduce cycle time!

deployment
The Build Process
Source Code ➔ Build System ➔ Source Code

Build System
# KERNELRELEASE can change from a few different places, meaning version.h
# needs to be updated, so this check is forced on all builds

uts_len := 64
define filechk_utsrelease.h
    if [ `echo -n "$(KERNELRELEASE)" | wc -c ` `gt $(uts_len) ]; then \
        echo ""$(KERNELRELEASE)" exceeds $(uts_len) characters' >&2; \
        exit 1;
        fi;
    (echo \#define UTS_RELEASE "$(KERNELRELEASE)\";)
endef
define filechk_version.h
    (echo \#define LINUX_VERSION_CODE $(shell \
        expr $(VERSION) \* 65536 + $(PATCHLEVEL) \* 256 + $(SUBLEVEL)); \
        echo \#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c));)
endef

include/linux/version.h: $(srctree)/Makefile FORCE
    $(call filechk,version.h)

include/generated/utsrelease.h: include/config/kernel.release FORCE
    $(call filechk,utsrelease.h)

PHONY += headerdep
headerdep:
    $(Q)find include/ -name '.*.h' | xargs --max-args 1 scripts/headerdep.pl
Build Systems are Complex

Our record so far is a project we inherited with an Ant script weighing in at 10,000 lines of XML. Needless to say, this project required an entire team devoted to keeping the build working—a complete waste of resources. [Jez Humble & David Farley]

KDE 4 is leaving the aging "autotool" build chain behind. Some developers, not only in KDE, like to nickname the autotools as "auto-hell" because of its difficulty to comprehend architecture. [http://lwn.net/Articles/188693/]

I hate GNU Make especially those Makefiles that are hand-crafted and undocumented. It all feels like a bad hack.

[Dean Berris]
>5 MLOC & ~20 Years of History
Quake 3

server
game logic

quake3.exe

network
d(a)emon

client UI
client renderer

do

16
Build Systems Grow in Complexity
Build Systems Require 12% of a Developer’s Time (on average)

Build maintenance slows development!

Kumfert, G., and Epperly, T. 
*Software in the DOE: The Hidden Overhead of the “Build”*
>36 MLOC & ~120 Years of History
Build Files Change Relatively More than Source Code Files

- #changed source lines/source file
- #changed build lines/build file
The Build System Requires Significant Maintenance

- Source ⇒ Build
- Test ⇒ Build

12%
The Build System Requires Significant Maintenance

Low coupling thanks to higher-level build abstraction
Our Build Systems need HELP
Release Engineering

in-house/3rd party development

integrate

build

test

integration

reduce cycle time!

deployment
The Integration Process
Integrators are Gatekeepers
Integrators are Gatekeepers.
>180k Packages & ~28 Years of History

debian

FreeBSD®

ubuntu
Mozilla Delivers **New Version** of Firefox – First Web Browser to Support Do Not Track on Multiple Platforms! [Mozilla Blog]
Mozilla Delivers **New Version** of Firefox – First Web Browser to Support Do Not Track on Multiple Platforms!

[Mozilla Blog]
Risk Analysis & Cherry-picking

Upstream Sync
Risk Analysis & Cherry-picking

UNIX diff
Risk Analysis & Cherry-picking

Upstream Sync

UNIX diff

maintainer
Risk Analysis & Cherry-picking

Upstream Sync

UNIX diff

maintainer
Library Transitions Ripple through the whole System

"Macro-level software evolution: a case study of a large software compilation" (Gonzalez-Barahona et al.)
Library Transitions Ripple through the whole System

Dependency Management

Fig. 7 Most popular instance of the Inter-Dependency Graph for mozilla in Debian 2.0. Each of the two abstract dependencies have only one child.

In Debian 2.0 the packages with more dependencies had 19 (python-gdk-imlib, boot-floppies, and libhdf4-altdev). In Debian 4.0 the package with the largest number of dependencies is kde, with 561, followed by gnome, with 486. kde and gnome are sets of GUI applications for the Unix desktop, none of them is present in Debian 2.0. Both kde and gnome are bundles of packages. In practical terms this means that they do not have any source code associated: when these packages are installed, the bundle is installed.

As the number of dependencies of packages evolves, their dependency graphs are likely to change too. For example, Fig. 6 shows the pIDG of mozilla in Debian 4.0, which can be compared to its dependency graph in Debian 2.2, depicted in Fig. 7.

Mozilla required 13 packages in 2.2 (the first version of Debian to include it), and 72 in 4.0. This growth is expected as applications evolve and grow to satisfy newer requirements.

Table 7: Evolution of the number of all dependencies for some selected binary packages, for the studied Debian releases

<table>
<thead>
<tr>
<th>Package</th>
<th>Debian 2.0</th>
<th>Debian 2.1</th>
<th>Debian 2.2</th>
<th>Debian 3.0</th>
<th>Debian 3.1</th>
<th>Debian 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache1.3</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>45</td>
<td>57</td>
<td>64</td>
</tr>
<tr>
<td>Mozilla</td>
<td>N/A</td>
<td>7</td>
<td>13</td>
<td>21</td>
<td>75</td>
<td>72</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>9</td>
<td>16</td>
<td>9</td>
<td>23</td>
<td>54</td>
<td>42</td>
</tr>
</tbody>
</table>

"Macro-level software evolution: a case study of a large software compilation" (Gonzalez-Barahona et al.)
Library Transitions Ripple through the whole System

Dependency Management

Fig. 7

Most popular instance of the Inter-Dependency Graph for mozilla in Debian 2.2. Each of the two abstract dependencies have only one child

In Debian 2.0 the packages with more dependencies had 19 (python-gdk-imlib, boot-floppies, and libhdf4-altdev). In Debian 4.0 the package with the largest number of dependencies is kde, with 561, followed by gnome, with 486. kde and gnome are sets of GUI applications for the Unix desktop, none of them is present in Debian 2.0. Both kde and gnome are bundles of packages. In practical terms this means that they do not have any source code associated: when these packages are installed, the bundle is installed. This raises three noteworthy issues: first, from the point of view of the user installing such bundles, these collections of packages operate as a single software product; second, it can be argued that these packages inflate the average number of dependencies without adding any new source code themselves; and third, they can be considered a great demonstration of the power of component-oriented software engineering, where a "new" application, the bundle, can be created from many components without writing a single line of code.

As the number of dependencies of packages evolves, their dependency graphs are likely to change too. For example, Fig. 6 shows the pIDG of mozilla in Debian 4.0, which can be compared to its dependency graph in Debian 2.2, depicted in Fig. 7.

Mozilla required 13 packages in 2.2 (the first version of Debian to include it), and 72 in 4.0. This growth is expected as applications evolve and grow to satisfy newer requirements.

Table 7

<table>
<thead>
<tr>
<th>Package</th>
<th>2.0</th>
<th>2.1</th>
<th>2.2</th>
<th>3.0</th>
<th>3.1</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1.3</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>45</td>
<td>57</td>
<td>64</td>
</tr>
<tr>
<td>Mozilla</td>
<td>N/A</td>
<td>7</td>
<td>13</td>
<td>21</td>
<td>75</td>
<td>72</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>9</td>
<td>16</td>
<td>9</td>
<td>23</td>
<td>54</td>
<td>42</td>
</tr>
</tbody>
</table>

"Macro-level software evolution: a case study of a large software compilation" (Gonzalez-Barahona et al.)
Library Transitions Ripple through the whole System

"Macro-level software evolution: a case study of a large software compilation" (Gonzalez-Barahona et al.)
2-way Impact Analysis

Did someone break my package?

Mozilla required 13 packages in 2.2 (the first version of Debian to include it), and 72 in 4.0. This growth is expected as applications evolve and grow to satisfy newer requirements.
Figure 7: Most popular instance of the Inter-Dependency Graph for `mozilla` in Debian 2.2. Each of the two abstract dependencies have only one child.

In Debian 2.0 the packages with more dependencies had 19 (e.g., `python-gdk-imlib`, `boot-floppies`, and `libhdf4-altdev`). In Debian 4.0 the package with the largest number of dependencies is `kde`, with 561, followed by `gnome`, with 486. `kde` and `gnome` are sets of GUI applications for the Unix desktop, none of them is present in Debian 2.0. Both `kde` and `gnome` are bundles of packages. In practical terms this means that they do not have any source code associated: when these packages are installed, the bundle is installed. This raises three noteworthy issues: first, from the point of view of the user installing such bundles, these collections of packages operate as a single software product; second, it can be argued that these packages inflate the average number of dependencies without adding any new source code themselves; and third, they can be considered a great demonstration of the power of component-oriented software engineering, where a "new" application, the bundle, can be created from many components without writing a single line of code.

As the number of dependencies of packages evolves, their dependency graphs are likely to change too. For example, Figure 6 shows the PIDG of `mozilla` in Debian 4.0, which can be compared to its dependency graph in Debian 2.2, depicted in Figure 7. Mozilla required 13 packages in 2.2 (the first version of Debian to include it), and 72 in 4.0. This growth is expected as applications evolve and grow to satisfy newer requirements.

Table 7: Evolution of the number of all dependencies for some selected binary packages, for the studied Debian releases.

<table>
<thead>
<tr>
<th>Package</th>
<th>2.0</th>
<th>2.1</th>
<th>2.2</th>
<th>3.0</th>
<th>3.1</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache1.3</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>45</td>
<td>57</td>
<td>64</td>
</tr>
<tr>
<td>Mozilla</td>
<td>N/A</td>
<td>7</td>
<td>13</td>
<td>21</td>
<td>75</td>
<td>72</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>9</td>
<td>16</td>
<td>9</td>
<td>23</td>
<td>54</td>
<td>42</td>
</tr>
</tbody>
</table>

Did I break someone's package?

Did someone break my package?
Luciano Bello discovered that the random number generator in Debian's openssl package is predictable. This is caused by an incorrect Debian-specific change to the openssl package (CVE-2008-0166). As a result, cryptographic key material may be guessable.

It is strongly recommended that all cryptographic key material which has been generated by OpenSSL versions starting with 0.9.8c-1 on Debian systems is recreated from scratch. Furthermore, all DSA keys ever used on affected Debian systems for signing or authentication purposes should be considered compromised; the Digital Signature Algorithm relies on a secret random value used during signature generation.

The first vulnerable version, 0.9.8c-1, was uploaded to the unstable distribution on 2006-09-17, and has since propagated to the testing and current stable (etch) distributions. The old stable distribution (sarge) is not affected.
Luciano Bello discovered that the random number generator in Debian's openssl package is predictable. This is caused by an incorrect Debian-specific change to the openssl package (CVE-2008-0166). As a result, cryptographic key material may be guessable.

It is strongly recommended that all cryptographic key material which has been generated by OpenSSL versions starting with 0.9.8c-1 on Debian systems is recreated from scratch. Furthermore, all DSA keys ever used on affected Debian systems for signing or authentication purposes should be considered compromised; the Digital Signature Algorithm relies on a secret random value used during signature generation.

The first vulnerable version, 0.9.8c-1, was uploaded to the unstable distribution on 2006-09-17, and has since propagated to the testing and current stable (etch) distributions. The old stable distribution (sarge) is not affected.

Local Patch

>=1.5 years 8-(
Luciano Bello discovered that the random number generator in Debian's openssl package is predictable. This is caused by an incorrect Debian-specific change to the openssl package (CVE-2008-0166). As a result, cryptographic key material may be guessable.

It is strongly recommended that all cryptographic key material which has been generated by OpenSSL versions starting with 0.9.8c-1 on Debian systems is recreated from scratch. Furthermore, all DSA keys ever used on affected Debian systems for signing or authentication purposes should be considered compromised; the Digital Signature Algorithm relies on a secret random value used during signature generation.

The first vulnerable version, 0.9.8c-1, was uploaded to the unstable distribution on 2006-09-17, and has since propagated to the testing and current stable (etch) distributions. The old stable distribution (sarge) is not affected.
Open Challenges
Improving Software and Build Comprehension
Where are the Tools?

- refactor your makefiles!
- test your build!
- what did the other team break now ;-) 
- keep poking those upstream guys until they give in :-)

eclipse
RELEASE IDE
On average we deploy new code **fifty** times a day.
On average we deploy new code **fifty** times a day.
On average we deploy new code fifty times a day.

Release Engineering

Integration in-house/3rd party development

keep cycle time down

integrates

test

build

deployment
On average we deploy new code fifty times a day.

- in-house/3rd party development
- integrate
- test
- build

keeping cycle time down

deployment
On average we deploy new code fifty times a day.

The Build System Requires Significant Maintenance

- Source ⇒ Build
- Test ⇒ Build

<table>
<thead>
<tr>
<th>Source ⇒ Build</th>
<th>Test ⇒ Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>27%</td>
<td>44%</td>
</tr>
<tr>
<td>16%</td>
<td>20%</td>
</tr>
<tr>
<td>4%</td>
<td>8%</td>
</tr>
</tbody>
</table>

- mozilla
- eclipse
- jazz
On average we deploy new code **fifty** times a day.

The Build System Requires Significant **Maintenance**

![Chart showing maintenance percentages for various software projects such as Mozilla, Eclipse, and Jazz.](https://behrns.files.wordpress.com/2008/03/ikea-car.jpg)
On average we deploy new code fifty times a day.

Release Engineering deployment in-house/3rd party development

The Build System Requires Significant Maintenance

Risk Analysis & Cherry-picking

Upstream Sync

maintainer
On average we deploy new code **fifty** times a day.

Release Engineering

- in-house/3rd party development
- integrate
- test
- build
- deployment

keeping cycle time down

The Build System Requires **Significant Maintenance**

<table>
<thead>
<tr>
<th>Source → Build</th>
<th>Test → Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>27%</td>
<td>16%</td>
</tr>
<tr>
<td>44%</td>
<td>20%</td>
</tr>
<tr>
<td>12%</td>
<td>4%</td>
</tr>
<tr>
<td>8%</td>
<td>4%</td>
</tr>
</tbody>
</table>

Risk Analysis & Cherry-picking

- Upstream Sync
- diff
- maintainer

Mozilla
- Eclipse
- Jazz