1001 Bugs – oder: die goldenen Regeln für schlechte Programmierung

Christian Boltz
openSUSE beta tester, PostfixAdmin developer,
battle-hardened AppArmorer, …
and: BBfH

Verwende nie Bibliotheken oder existierende Funktionen

Das Rad neu erfinden macht Spaß!

```
function myprint ($text) {
   $handle = fopen("/dev/stdout");
   fput($handle, $text);
}
```

Photo: http://www.flickr.com/photos/vrogy/514733529/

Spezielle Werte verdienen eine besondere Behandlung

looks like you have some special code in yast for password "x", maybe I should use the even more secure new password "y" in the future ?! ;-)

[Harald Koenig, bnc#148464]

Erfinde neue Wege, um ein Programm langsam zu machen

```
while ( $current < $list['alias_count'] ) {
    $query = "SELECT $table_alias.address FROM $table_alias
        [...] LIMIT $current, 1";
    $result = db_query ("$query");
    $row = db_array ($result['result']);
    $tmpstr = $row['address'];
    $idxlabel = $tmpstr[0] . $tmpstr[1]; // first two chars
    $current = $current + $page_size;
    $pagebrowser[]=$idxlabel;
}
```

Erfinde neue Wege, um ein Programm langsam zu machen

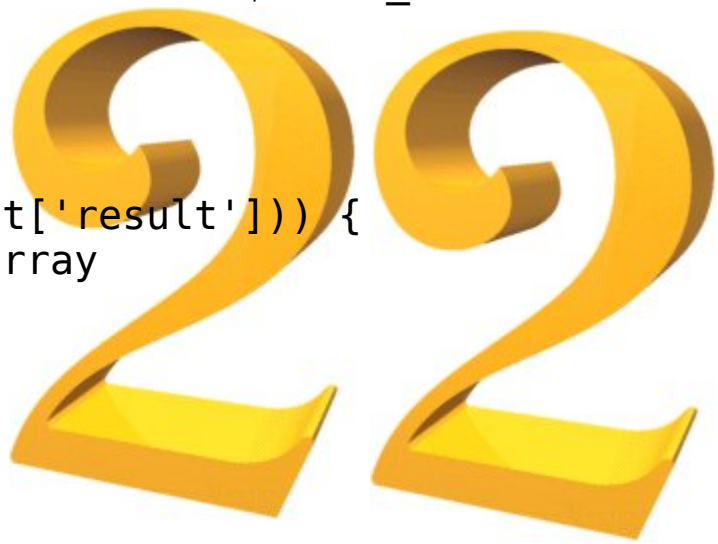
```
$initcount = "SET @row=-1";
$result = db_query($initcount);

# get labels for relevant rows (first and last of each page)
$page_size_zerobase = $page_size - 1;

$query = "
    SELECT * FROM (
        SELECT $idxfield AS label,
        @row := @row + 1 AS row $querypart
    ) idx WHERE MOD(idx.row, $page_size)
    IN (0,$page_size_zerobase) OR idx.row = $count_results
";

$result = db_query ($query);

if ($result['rows'] > 0) {
    while ($row = db_array ($result['result'])) {
        # store all labels in an array
    }
}
```



Benutzer hassen Fehlermeldungen

Schlussfolgerung: Zeige nie eine Fehlermeldung an, auch wenn irgendwas schiefgeht.

openSUSE-Entwickler scheinen diese Regel zu mögen:

- no error message when RPM database is missing (bnc#148105)
- rcxdm doesn't start X in failsafe mode if xorg.conf.install was deleted (bnc#394316)

kweather nicht installiert – und jetzt?

> Hmm, what about looking out of the window?

Outside of the window is another window, the desktop background or the border of the monitor. How exactly would that help?

[komplette Story: bnc#141107]

Gib niemals Rechte ab

... Du könntest sie später nochmal brauchen

- aa-notify hat in openSUSE 11.4 wegen fehlender Berechtigungen nicht funktioniert
- gilt auch für die “real world”

Mache das UI einfach verständlich

[bnc#218677]

Mache das UI einfach verständlich

vi-Befehle sind sogar relativ einfach zu merken.

Wenn man einmal weiß, was
dw db de d) d(d} d{ dd d^ d$ d0 dG sowie
cw und yw machen, dann weiß man auch, was
cb ce c) c(c} c{ cc c^ c$ c0 cG sowie
yb ye y) y(y} y{ yy y^ y$ y0 yG machen.

[Bernd Brodesser in suse-linux]

Wähle möglichst schlechte Voreinstellungen

`zypper ar obs://home:cboltz home:cboltz`

fügt immer das Factory-Repository hinzu, nicht das Repositiory für die installierte Distribution.

Es gibt aber eine Config-Option zum Festlegen der Distribution in zypper.conf (und um nach einem Distributionsupdate Spaß zu haben)

(wie wärs mit /etc/SuSE-release?)

[bnc#648892]

[wontfix, ENOTIME]

Wähle möglichst schlechte Voreinstellungen

Don't tell people to edit their polkit privileges to individual needs.

Make a usable system, or at least expose a big and visible button saying "make this system usable".

[Linus Torvalds, bnc#731812]

Es reicht, die üblichen Daten zu erwarten

(cron.daily läuft nicht, weil das System angeblich im Batteriebetrieb läuft)

Yes Karl, your machine has a battery! But no ac adapter :-)

Unfortunately it is the battery in the BT Mouse and it won't be able to power your system for very long :-)

[Stefan Seyfried, bnc#221999]

Es reicht, die üblichen Daten zu erwarten

```
#define IM_TEXT_LEN      32
char numstr[20];
char str[IM_TEXT_LEN];
sprintf(numstr, "%%2d%%%% %%.%ds",
    IM_TEXT_LEN-6);

sprintf(str, numstr,
    (int)(percent * 100),
    graph->pairs[i]->name);
```

[modlogan, bnc#517602]

Es reicht, die üblichen Daten zu erwarten

```
#define IM_TEXT_LEN      32
char numstr[20];
char str[IM_TEXT_LEN];
sprintf(numstr, "%%2d%%%% %%.%ds",
    IM_TEXT_LEN-6);
# numstr = "%2d%% %.27s"
sprintf(str, numstr,
    (int)(percent * 100),
    graph->pairs[i]->name);
```

[modlogan, bnc#517602]

Mach Deinen Code niemals wiederverwendbar

... vor allem wenn er mehr als 1000 Zeilen hat

Code wiederverwenden ist wie Sätze aus Texten anderer Leute zu nehmen und zu versuchen, einen Magazin-Artikel daraus zu machen.
[Bob Frankston, übersetzt]

Davon abgesehen:

Niemand braucht soviel Code ein zweites Mal ;-)

Mach ALL Deinen Code wiederverwendbar

(auch jedes kleine Script, das Du schreibst)

schlecht:

```
echo "Hello World!";
```

Mach ALL Deinen Code wiederverwendbar

gut:

```
Class HelloWorld {
    my $greeting = "Hello World!";
    function setGreet($newGreeting) {
        $greeting = $newGreeting;
    }
    function greet() {
        echo $greeting;
    }
}
$greeter = new HelloWorld;
# default text is fine
$greeter->greet;
```

Mach ALL Deinen Code wiederverwendbar

Das spart Dir in Zukunft einen Haufen Arbeit.
Du kannst dann einfach folgendes machen:

```
$greeter = new HelloWorld;
$greeter->setGreet("Hello openSUSE!");
$greeter->greet;
```

Mach ALL Deinen Code wiederverwendbar

Das spart Dir in Zukunft einen Haufen Arbeit.
Du kannst dann einfach folgendes machen:

```
$greeter = new HelloWorld;
$greeter->setGreet("Hello openSUSE!");
$greeter->greet;
```

Das ist viel einfacher als

```
echo "Hello openSUSE!";
```

Verwende keine if-Blöcke.
Verwende lieber cp.

```
cp edit-mailbox.php admin/edit-mailbox.php
vi admin/edit-mailbox.php # remove permission checks
diff edit-mailbox.php admin/edit-mailbox.php | wc -l
    15
... 5 Jahre später ...
diff edit-mailbox.php admin/edit-mailbox.php | wc -l
    250
```

Verwende keine if-Blöcke. Verwende lieber cp.

Langweilig:

```
[Unit]
Description=Daemon to detect crashing apps
After=syslog.target

[Service]
ExecStart=/usr/sbin/abrtd
Type=forking

[Install]
WantedBy=multi-user.target
```

Verwende keine if-Blöcke. Verwende lieber cp.

Genauso langweilig:

CPAN_service im openSUSE Build Service

Das würde es zu deutlich machen, dass Du als Packager ein Faulpelz bist und ein Specfile aus einer Vorlage verwendest.

Die Paketpflege macht viel mehr Spaß, wenn Du mit cpanspec generierte Specfiles eincheckst.

Traue Deinen Benutzern
oder: überprüfe Benutzereingaben nie

Sogar AppArmor hielt sich an diese Regel, sie kann also nicht zu falsch sein ;-)

```
echo 'AAA AAA' > /proc/$$/attr/current
Segmentation fault
```

[107353.169142] kernel BUG at kernel BUG at /usr/src/packages/BUILD/kernel-desktop-2.6.37.6/linux-2.6.37/security/apparmor/audit.c:183!

[107353.169159] invalid opcode: 0000 [#7] SMP

[...]

[https://bugs.launchpad.net/bugs/789409]

Traue Deinen Benutzern
oder: überprüfe Benutzereingaben nie

http://server/wbb/images/avatars/myphoto.php

bereitet dem Admin etwas[tm] Spaß...

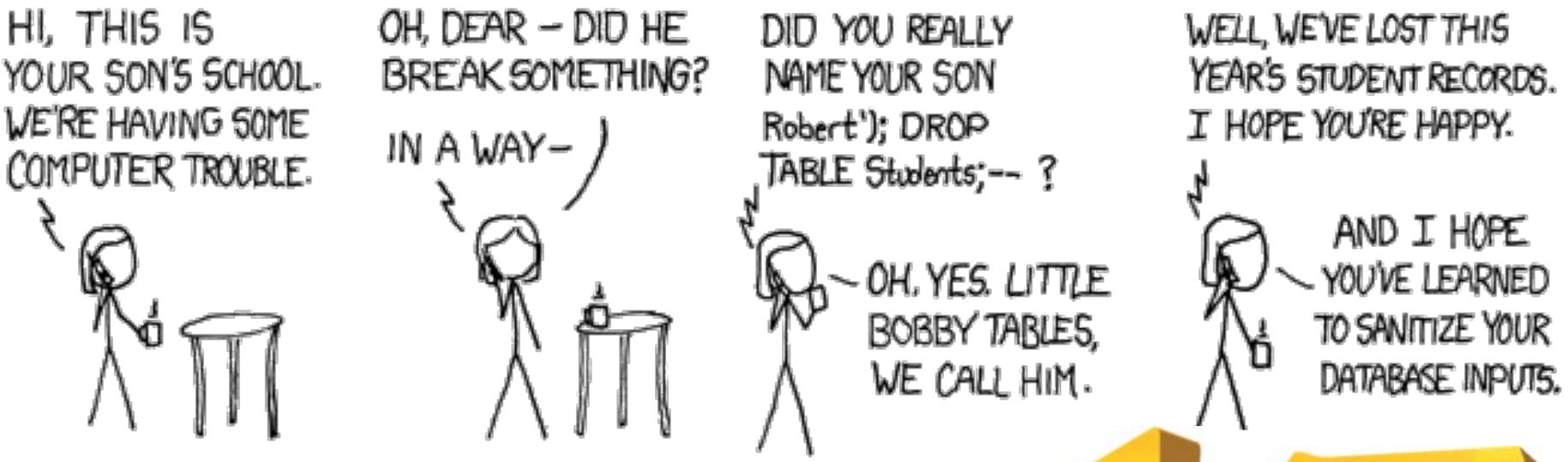
[Woltlab burning board 1.0.2]

Traue Deinen Benutzern
oder: überprüfe Benutzereingaben nie

Der von Ihnen vielleicht erwartete Input wird zu dem eines verstimmten Mitarbeiters oder eines Crackers, der Monate Zeit hat, oder einer Katze, die über die Tastatur läuft, in keinerlei Zusammenhang stehen.

[http://www.php.net/manual/de/security.general.php]

Traue Deinen Benutzern
oder: überprüfe Benutzereingaben nie



[http://xkcd.com/327/]

Prüfe immer auf Fehler

Vor allem, wenn Du eine Bibliothek schreibst.

Es ist eben nicht trivial
korrekten Code zu schreiben.

```
exit(-1);
syslog(L0G_ERROR, “Can't exit.\n”);
```

[Lutz Donnerhacke in dclp]

Denke nie an Fehlerbehandlung

Erwarte einfach, dass Dein Code funktioniert.

- Fehlerbehandlung ist nur langweilige Programmierarbeit. Es ist viel aufregender, Fehler nach dem Deployen auf wichtigen Systemen zu fixen
- Investiere Deine Zeit lieber in die Entwicklung neuer Features
- davon abgesehen ist das Nachdenken über Fehler sowieso die Aufgabe von Bugreportern

Bugzilla-Geschwindigkeitsvergleich mit Coolo

> Status?
NEW
[Ihno Krumreich and Stephan Kulow, bnc#159223]

> which camera is this?
Marcus, this is my bug :)
[Marcus Meissner and Stephan Kulow, bnc#217731]

Erwarte nie, dass jemand Deine Software nutzt

Schreib ruhig Deinen Benutzernamen hartcodiert rein.

> mkdir: Can't create directory »»/home/ratti««

> Du solltest lieber ~ statt
> /home/ratti verwenden ;-)

Warte nur ab, die 0.0.3 funktioniert sogar, wenn man nicht "ratti" heisst. ;-)

[Spaß mit Ratti bei der Fontlinge-Entwicklung]

Erwarte nie, dass jemand Deine Software nutzt

Entferne niemals Debugging-Code
(zumindest solange Dich keiner dazu zwingt)

mcelog should NOT email trenn@suse.de by default

[bnc#713562]

Erwarte nie, dass jemand Deine Software nutzt

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

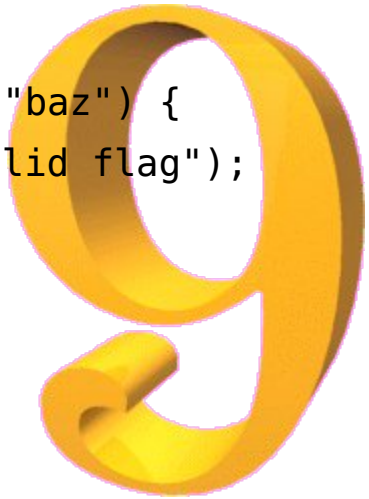
[Linus Torwalds, August 1991]

Verschachtle Deinen Code so tief wie möglich

```
if ($name == “foo”) {
    if ($value == “bar”) {
        if ($number > 0) {
            if ($flags == "baz") {
                # 1000 Zeilen Code
            } else {
                die ("invalid flag");
            }
        } else {
            die ("invalid number");
        }
    } else {
        die ("invalid value");
    }
} else {
    die ("invalid name");
}
```

Mach niemals sowas:

```
if ($name != “foo”) {
    die("invalid name");
}
if ($value != "bar") {
    die ("invalid value");
}
if ($number <= 0) {
    die("invalid number");
}
if ($flags != "baz") {
    die ("invalid flag");
}
# 1000
# Zeilen
# Code
```



Mach Gedächtnistraining mit Deinen Benutzern

```
# zypper ar --help |grep refresh
-f, --refresh     Enable autorefresh of the repository.
# zypper mr --help |grep refresh
-r, --refresh     Enable auto-refresh of the repository.
```

[bnc#661410]

Ignoriere Compiler- and rpmlint-Warnungen

- echte Probleme verursachen Fehler, keine Warnungen
- das heißt: Warnungen sind kein Problem

Reiche Deine Patches nie upstream ein

Es macht Spaß, alle Patches im Distributions-Paket zu behalten:

- Du siehst sehr professionell aus, wenn Du 50 Patches in einem Paket handhaben kannst
- Du sparst den upstream-Entwicklern die Arbeit, Patches zu reviewen und zu integrieren
- Du hast immer Spaß, wenn Du Dein Paket und Deine Patches auf die nächste Version aktualisiert

Schreibe niemals Dokumentation

- falls schon alte Doku existiert, aktualisiere sie auf keinen Fall
- Doku liest eh keiner
- Kommentare im Code zählen auch als Doku - vermeide sie nach Möglichkeit

Schreibe niemals Dokumentation

- Doku liest eh keiner

Wirklich?

Ich habe die 10.1 wie ein echter User getestet. In anderen Worten: Ich habe nie die Release Notes oder die Doku gelesen ;-)

[tomhorsley(at)adelphia.net
in opensuse-factory, übersetzt]

Schreibe niemals Dokumentation

- Doku liest eh keiner

Wirklich?
Vorsicht Büroklammern!

[bnc#65000]

Traue keinem Bugreporter

> > RESOLVED INVALID
> Henne, did you actually test this before closing
> the bug as invalid?
of course i did not test it. do you think i'm bored?

[bnc#420972]

Traue keinem Bugreporter

general rule: if Olaf reports a bug, it is a valid bug.
(Olaf Hering while reopening bnc#168595)

Spaß mit NEEDINFO

I am supposed to be the info provider,
so here is my answer:

42

By the way:

What is the question?

[Johannes Meixner, bnc#190173]

Teste nie kleine Änderungen

switch2nvidia:

  * fixed disabling Composite extension;
    script replaced "Option" with "Optioff" :-(

[commit message by Stefan Dirsch]

KMS_IN_INITRD="noyes"

[sed result, bnc#619218]

-ao=pulse,alsa

+,alsa

[setup-pulseaudio fun, bnc#681113]

Quoting in Shellscripten wird überbewertet

```
@@ -352,1 +352,1 @@
-   rm -rf /usr /lib/nvidia-current/xorg/xorg
+   rm -rf /usr/lib/nvidia-current/xorg/xorg
```

Wahrscheinlich der am Meisten kommentierte Commit auf github.

https://github.com/MrMEEE/bumblebee/commit/a047be

So wird man berühmt! ;-)

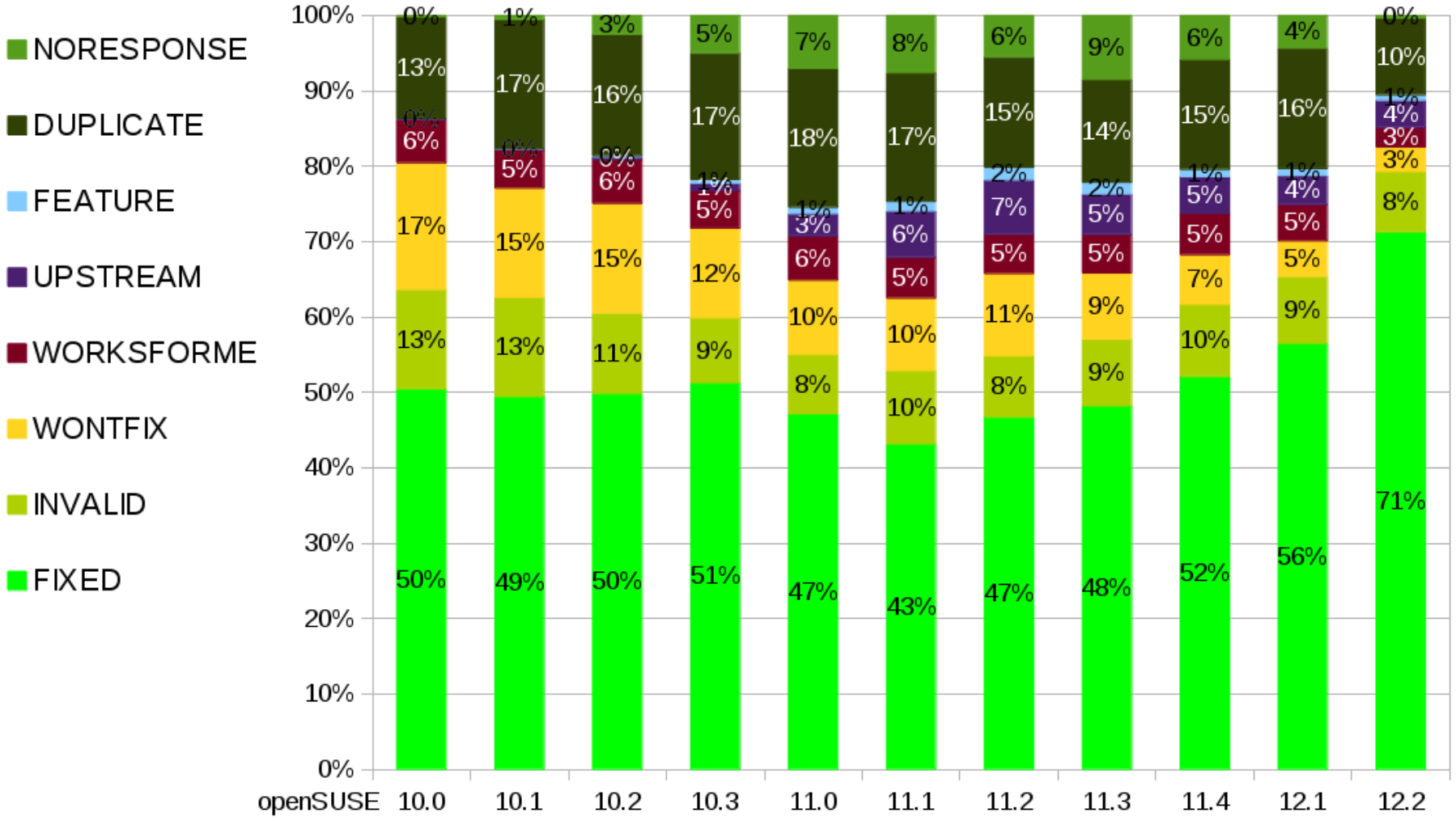
Bughandling in openSUSE

Wir werden auch nicht nach geschlossenen Bugs bezahlt.[1]

[1] Das hatten wir kürzlich schon mal und ich kann erst langsam wieder halbwegs fehlerfrei schreiben. Aber leicht schüttelt es mich immer noch vor Lachen.

[Lars Müller in opensuse-de]

“Bugreports bringen eh nichts!”



und der Gewinner ist…

Schreibe leicht verständlichen Code

```
#!/usr/bin/perl
eval eval '"'.

                  '#'.'!'.'/'.('['^'.')      .+(
              '['^'(').('['^')').'/'.('`'|'"').('`'|')')
          .('`'|'.').'/'.('['^'+').('`'|'%').('['^')'    ).
        ('`'|',').('!'^'+').('['^'+').('['^')').('`' |+ ( ((
     ')')))).('`'|'.').('['^'/').('{'^['].'\\'.'"'. ('`'  ^+
   '$').('`'|'!').('['^'(').('{'^['].('`'|')'). (((  ((  '['
  ))))^'(').('['^'/').('{'^['].('`'|',').((`')|    "\%").(
 '['^'(')   .('`'|'"').('`'|'!').('['^')').('`'|'%').(    '['
 ^')')            .('{'^['].('`'^'#').('`'|'/').('`'|'$').(
"\`"|   '%').'!'    .('{'^['].':'.'-'."\)".
'\\'   .((   '\\')   ).('`'         |"\.").
'\\'   .((     '"')   ).';'          .('!'
 ^'+'  ).'"';   $:=    '.'^           '~';
  ($~)         ='@'                    |+
   "\(";$^= (')')^
     '[';$/='`';
```

spezielle Regel für openSUSE:

spezielle Regel für openSUSE:

Programmiere, als ob der Kerl,
der irgendwann Deinen Code pflegt,
ein gewalttätiger Psychopath ist,
der weiß, wo Du wohnst.
[John F. Woods, übersetzt]

Danke!

- jedem, der versehentlich ;-) zu meinem Vortrag beigetragen hat
- für die Inspiration bei

  http://www.karzauninkat.com/Goldhtml/

  “Goldene Regeln für schlechtes HTML”

  http://www.sapdesignguild.org/community/design/golden_rules.asp

  “golden rules for bad user interfaces”

  http://blog.koehntopp.de/archives/2127-The-Importance-Of-FAIL.html

  http://blog.koehntopp.de/archives/2611-Was-bedeutet-eigentlich-Never-check-for-an-error-condition-you-dont-know-how-to-handle.html

  zwei großartige Artikel über FAIL von Kris Köhntopp
- perl Acme::EyeDrops fürs Rendern von Regel #1
- fürs Zuhören

Fragen?
Anmerkungen?
Flames?

Battle-hardened AppArmorer:by Sascha Peilicke

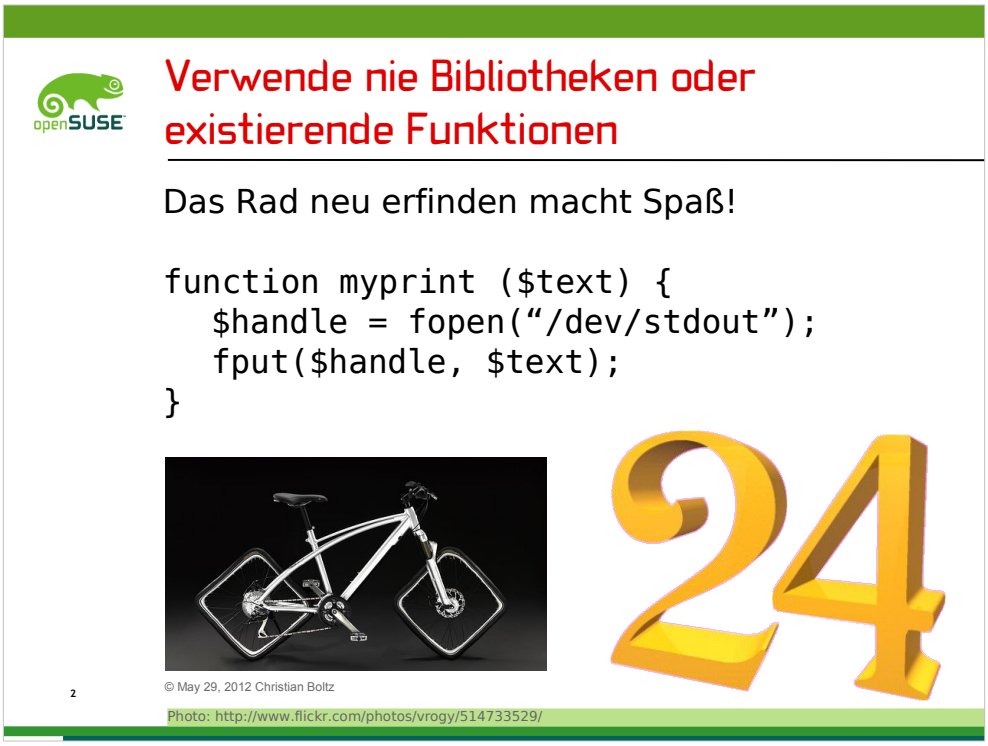
BBfH: Bastard Bugreporter from Hell

Man findet viele Bücher, die erklären, wie man guten Code schreibt.

Das ist nett und vielleicht sogar nützlich, aber langweilig ;-)

Dieser Vortrag bietet eine bessere Inspirationsquelle: Die goldenen Regeln für schlechte Programmierung.

Nebenbei: Ich habe keine Ahnung, warum Regeln immer golden sein müssen, aber ich werde nicht mit dieser Tradition brechen

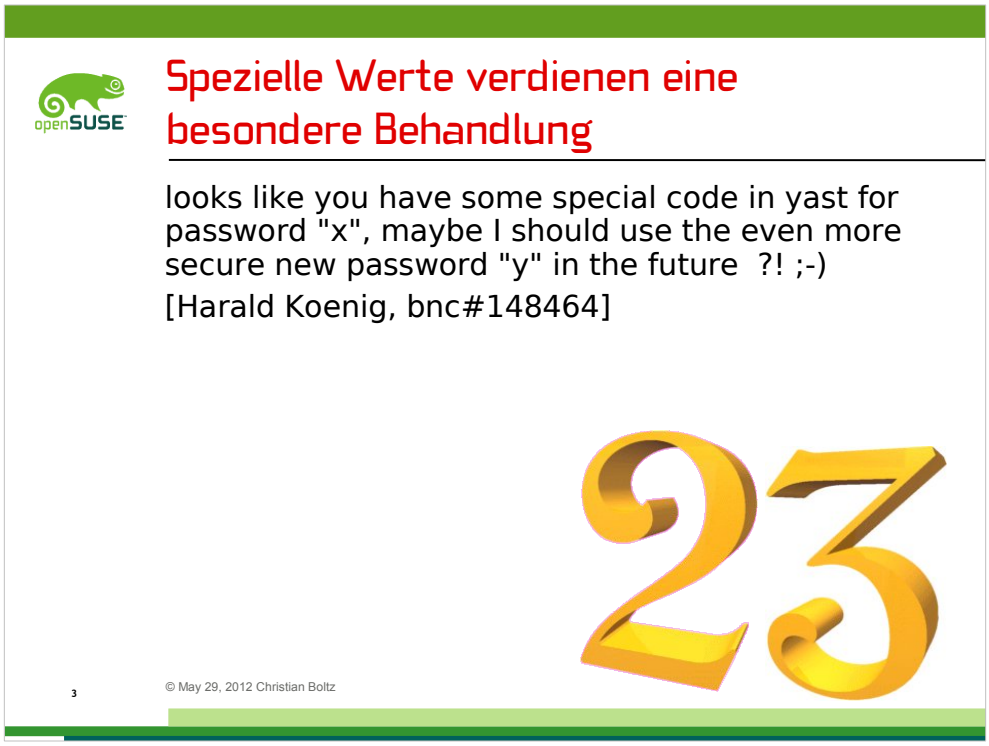


Die Verwendung von bestehenden Bibliotheken ist eine gaaanz schlechte Idee. Denkt doch nur mal an die Abhängigkeiten und den größeren des Platzbedarf, wenn Ihr viele Bibliotheken nutzt. Wollt Ihr das wirklich?

Ich muss zugeben, dass das Beispiel extrem ist, aber ich habe auch ein Praxisbeispiel – Befehlszeilenparameter parsen.

Am Anfang denkt man "sind doch nur zwei Optionen" - dafür braucht man keine Bibliothek. So ist es auch bei einem GsoC-Projekt im letzten Jahr passiert. Irgendwann ergänzt man dann mehr Befehlszeilenparameter, ein paar davon haben Optionen usw.

Irgendwann muss man dann doch zu getopts wechseln, das alles was man braucht kann

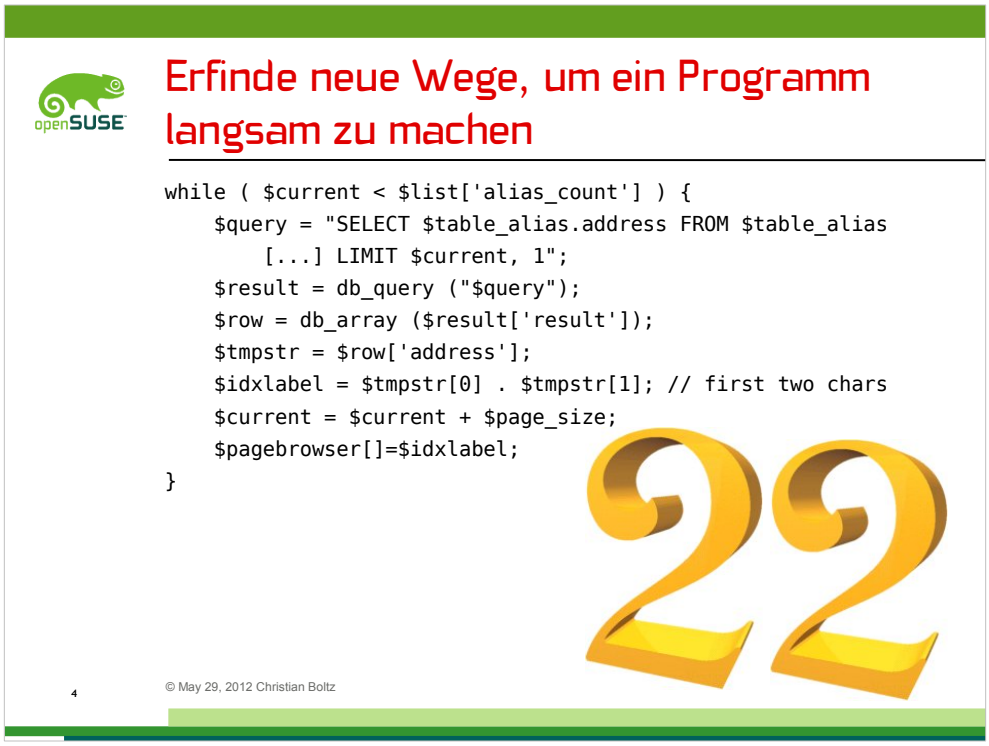


YaST behandelte das Passwort 'x' speziell und hat beim Anlegen eines Benutzers den Benutzer gleich gesperrt.

Es gibt mehr Beispiele, bei denen spezielle Werte auf besondere Weise behandelt wurden.

Denkt nur mal an die ganzen Jahr 2000-Bugs, bei denen das Jahr Null auf lustige Weise behandelt wurde.

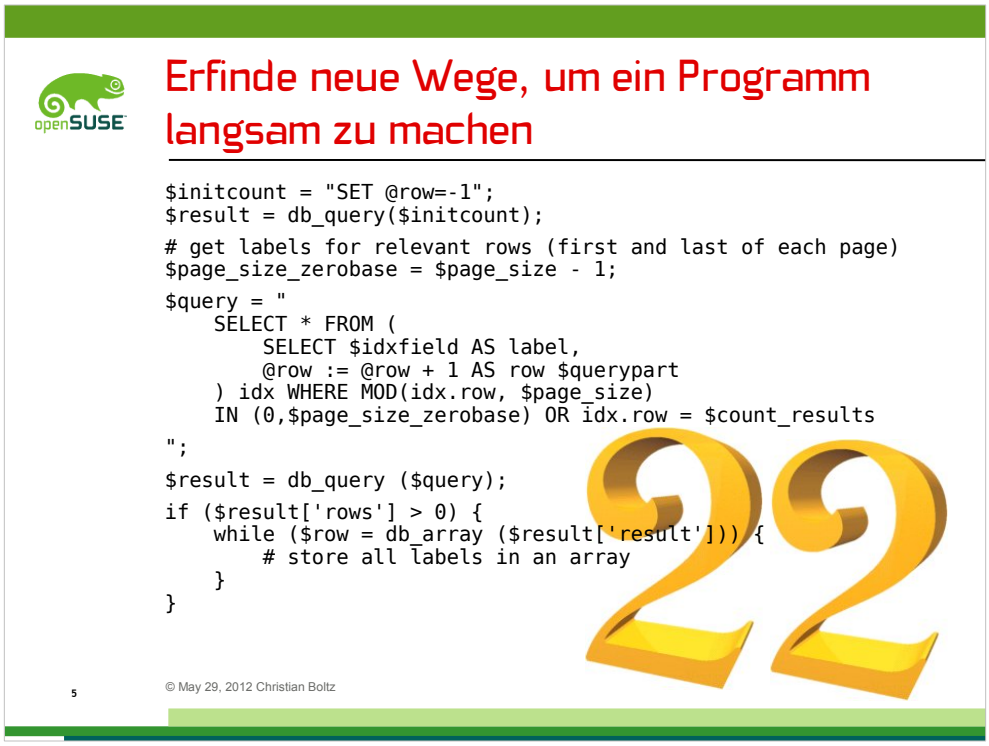
Linux bekommt 2038 ein vergleichbares Problem, wenn die Unix Time (Sekunden seit 1970) nicht mehr in eine 32bit-Variable passt. Ich bezweifle, dass dann noch jemand 32bit-Systeme benutzt, aber das Problem kann auch in Datenstrukturen, Dateiformaten oder Embedded-Geräten, z. B. eine Maschinensteuerung, versteckt sein



Eine sehr gute Methode, um Programme langsam zu machen, ist, SQL-Abfragen in einer Schleife zu machen.

Der Code ist ein verkürztes Schnipsel aus PostfixAdmin 2.3 und generiert den Pagebrowser (die “a-c, d-f, g-k” Links) in der Liste der Mailadressen. Der Code gezeigte Code liefert nur den Startpunkt jeder Seite, der Originalcode holt auch noch den Endpunkt ab, was die Anzahl der Abfrage verdoppelt.

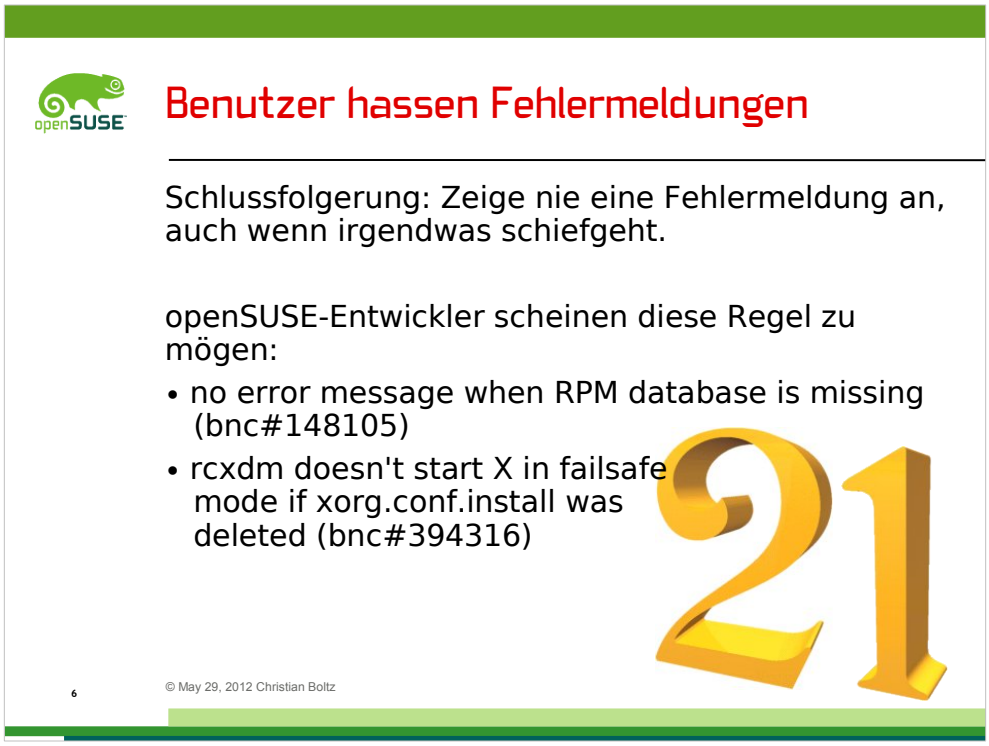
Die Einhaltung dieser Regel hat übrigens sehr gut funktioniert – mit 1000 Seiten Mailadressen hat es 10 Minuten gedauert, bis der Pagebrowser generiert war. Komischerweise haben sich sogar ein paar User über die Ladezeit dieser Seite beklagt...



Sowas sollte man nie tun, wenn man seine CPU beschäftigen will MySQL zählt die Zeilen durch und liefert als Ergebnis dieser riesigen Abfrage nur die relevanten Zeilen.

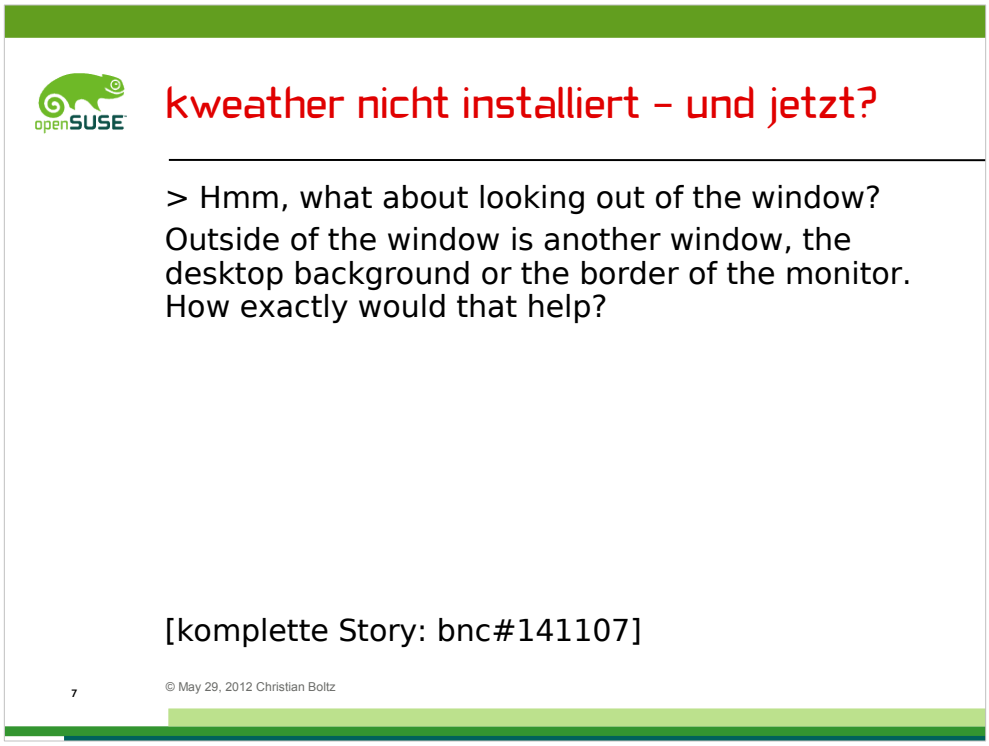
Der Nachteil ist, dass diese Abfrage nicht mit PostgreSQL funktioniert. Falls jemand eine funktionierende Lösung für PostgreSQL kennt – bitte nach dem Vortrag zu mir kommen

Oh, wenn wir schon bei Datenbanken sind: es gibt auch den guten alten Weg, ein Programm langsam zu machen. Erstellt einfach keinen Index in den Datenbank-Tabellen. Dioeses kleine Detail kann die Abfragen 100x langsamer machen.



Das ist die schwierigste Art Bugreports – der Versuch, einem Entwickler zu erklären, dass man gern eine Fehlermeldung hätte. Normalerweise ernet man eine Erklärung, warum das, was man getan hat, nicht funktionieren kann und dass man dafür unmöglich funktionierenden Code schreiben kann, und das Ganze sowieso ein “Corner Case” ist.

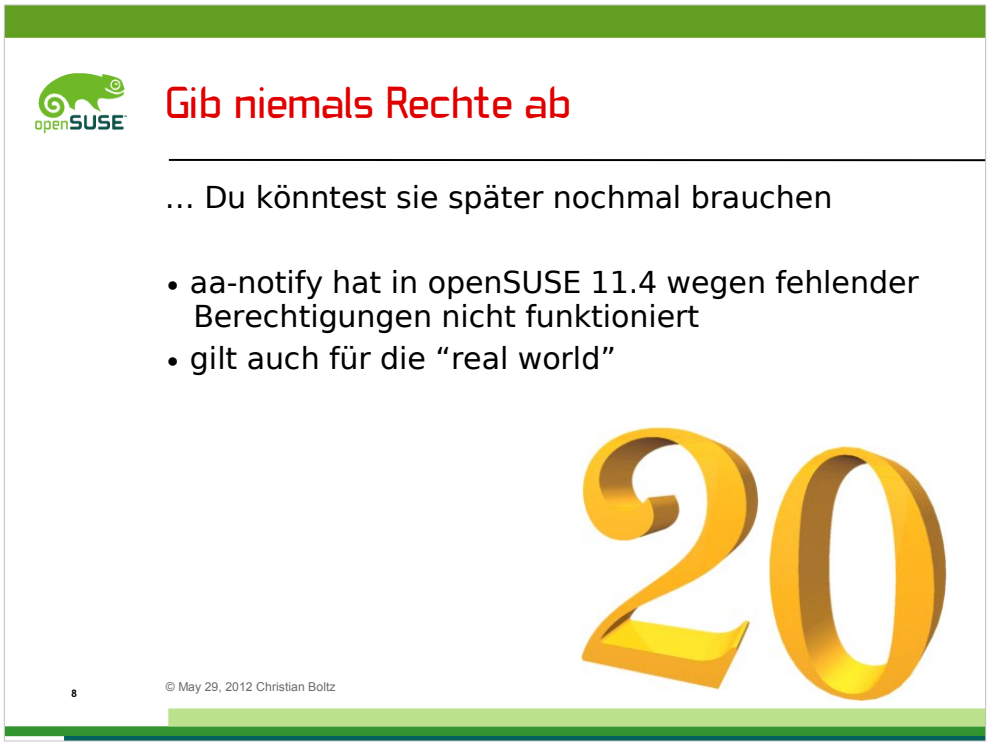
Etwa 5 Reopens später versteht dann jeder, dass man nur eine zusätzliche Fehlermeldung will. Das stand übrigens schon ganz oben im Bugreport...



Manchmal haben die Entwickler im SUSE-Büro wirklich schwierige Probleme zu lösen.

Eines Tages hat Marcus festgestellt, dass kweather nicht standardmäßig installiert wird. Ich habe dann vorgeschlagen, aus dem Fenster zu schauen.

Rasmus Plewe hat das dann probiert, aber nicht besonders erfolgreich

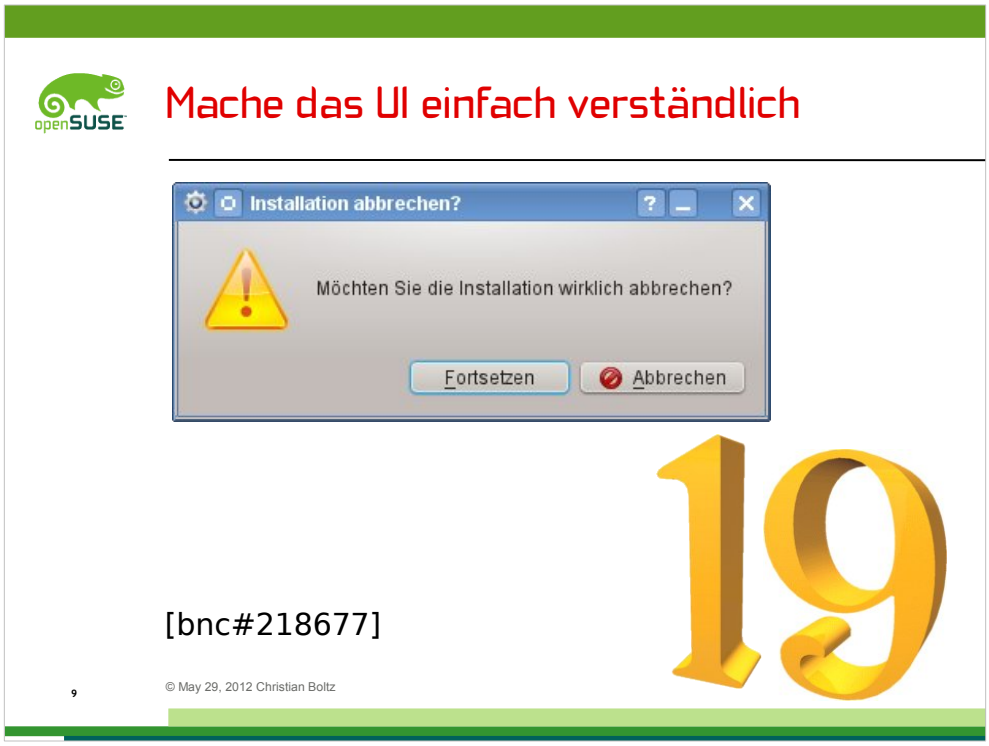


aa-notify ist ein Teil der AppArmor-Tools und kann eine Desktop Notification anzeigen, wenn ein Programm gegen sein AppArmor-Profil verstößt.

Durchs Abgeben der Berechtigungen konnte es /var/log/audit/ nicht mehr lesen.

Lustigerweise hat es die “Gib niemals Rechte ab”-Regel zumindest teilweise befolgt und hat immerhin die Gruppenrechte behalten. Dadurch hat es unter Ubuntu funktioniert, weil /var/log/audit/ dort für die Gruppe root lesbar ist – aber nicht auf openSUSE mit strengeren Berechtigungen.

Das ist der lustige Teil – manchmal heben sich zwei Bugs gegenseitig auf. Zumindest bis jemand wie ich kommt und sorgt mit den zu strengen Berechtigungen dafür, dass die nicht abgegebenen Gruppenrechte irgendwas nützen



Ratet mal, was passiertm, wenn man auf “Fortsetzen” klickt.

Die Antwort ist: YaST setzt den Abbruch der Installation fort – oder, um es für eventuell verwirrte Zuhörer zu übersetzen: Fortsetzen bricht die Installation ab.

Wenn man auf Abbrechen klickt, geht die Installation weiter.

Knöpfe mit “ja” und “nein” wären in diesem Fall eine gute Idee...

[[openSUSE 10.2]]

Mache das UI einfach verständlich

vi-Befehle sind sogar relativ einfach zu merken.

Wenn man einmal weiß, was
dw db de d) d(d} d{ dd d^ d$ d0 dG sowie
cw und yw machen, dann weiß man auch, was
cb ce c) c(c} c{ cc c^ c$ c0 cG sowie
yb ye y) y(y} y{ yy y^ y$ y0 yG machen.

[Bernd Brodesser in suse-linux]

Dieses Beispiel spricht für sich selbst – vi ist wirklich einfach zu benutzen

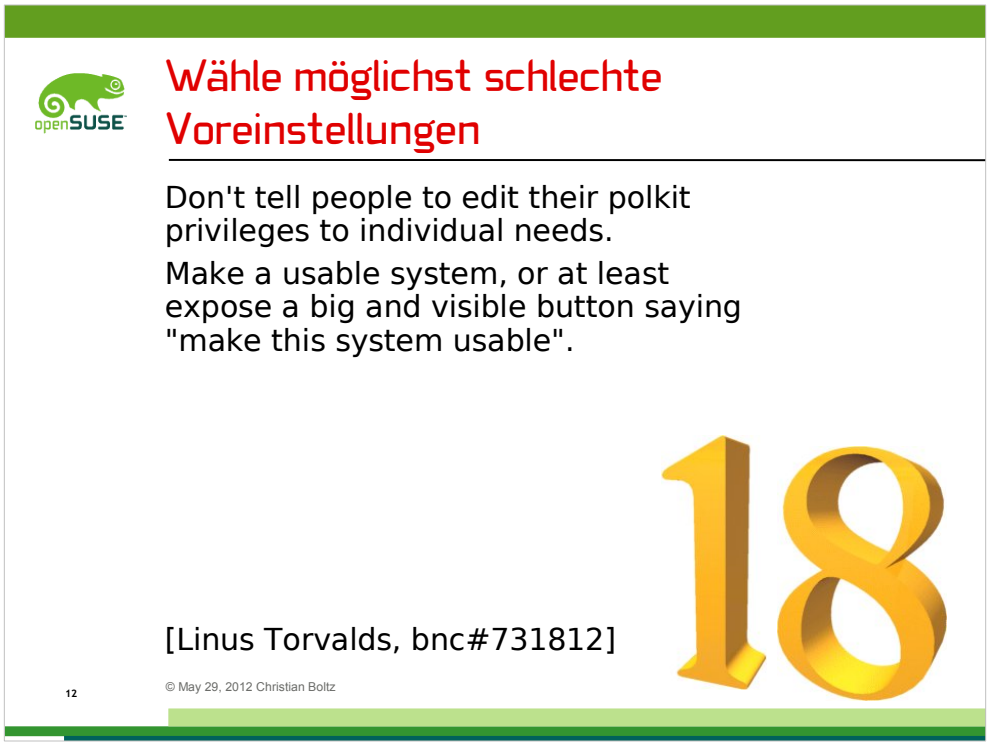
Wähle möglichst schlechte Voreinstellungen

`zypper ar obs://home:cboltz home:cboltz`
fügt immer das Factory-Repository hinzu, nicht das Repositiory für die installierte Distribution.

Es gibt aber eine Config-Option zum Festlegen der Distribution in zypper.conf (und um nach einem Distributionsupdate Spaß zu haben)

(wie wärs mit /etc/SuSE-release?)

[bnc#648892]
[wontfix, ENOTIME]



PolicyKit ist auch immer für Ärger zu haben – und beschert Regel 18 einen Promi-Bonus.

[[Kontext: Benutzer brauchten das root-Passwort, um WLAN-Verbindungen einzurichten.]]

[[Das Problem ist inzwischen übrigens gelöst]]

Es reicht, die üblichen Daten zu erwarten

(cron.daily läuft nicht, weil das System angeblich im Batteriebetrieb läuft)

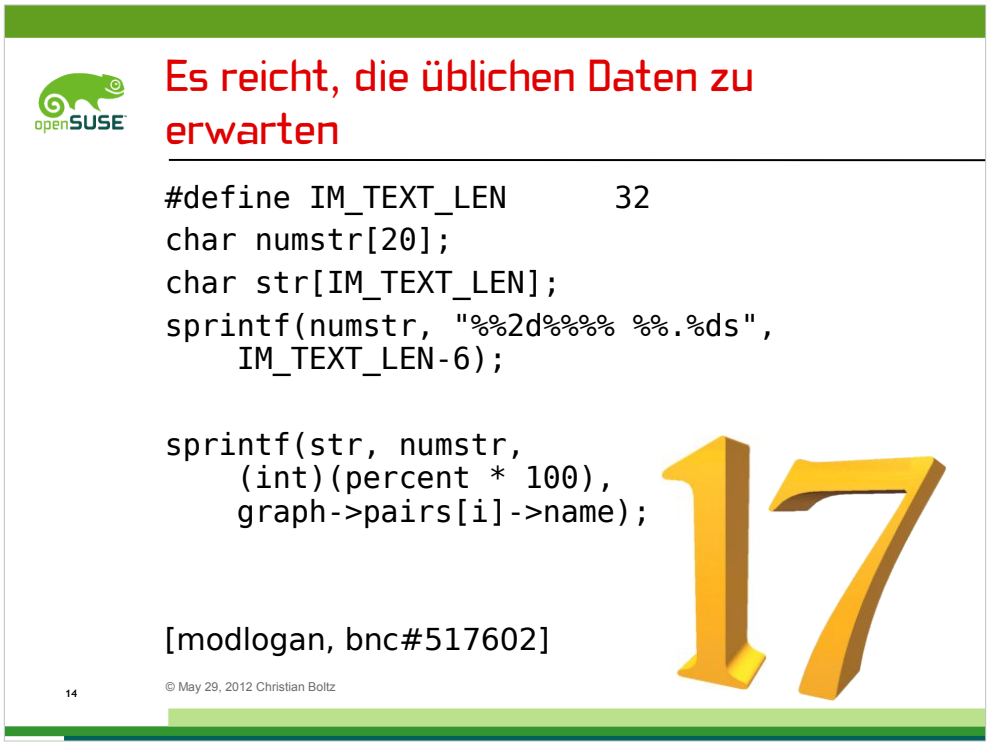
Yes Karl, your machine has a battery! But no ac adapter :-)

Unfortunately it is the battery in the BT Mouse and it won't be able to power your system for very long :-)

[Stefan Seyfried, bnc#221999]

Maus-Batterie wurde gefunden von

hal-find-by-capability --capability battery



Hier ist ein kleines Formatstring-Beispiel von modlogan – weiß jemand, was es macht?

Als erstes generiert es den Formatstring “numstr”

numstr: %2d → 2 Ziffern mit (int)(percent*100)

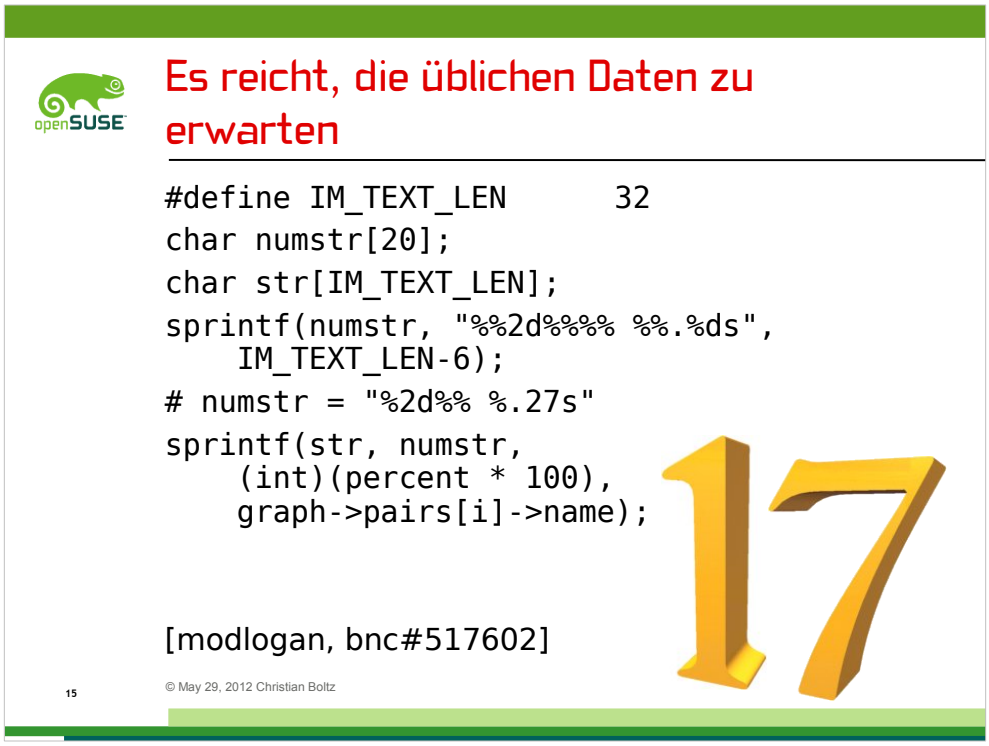
“%% “ → “%“ + Leerzeichen (2 byte)

%.27s → 27 Zeichen mit Punkten als Füllzeichen

insgesamt: 31 bytes

Und jetzt zur interessanten Frage: Warum und wie kann das einen Buffer Overflow auslösen?

Die Antwort: Mit nur einer Zeile im Log (also nur ein Client, nur ein Browsertyp) bekommt man 100%. Das sind 3 Ziffern, die dann den String ein Byte länger machen. Das gibt dann insgesamt 32 Byte und der Platz reicht nicht mehr fürs Nullbyte am Ende



Hier ist ein kleines Formatstring-Beispiel von modlogan – weiß jemand, was es macht?

Als erstes generiert es den Formatstring “numstr”

numstr: %2d → 2 Ziffern mit (int)(percent*100)

“%% “ → “%“ + Leerzeichen (2 byte)

%.27s → 27 Zeichen mit Punkten als Füllzeichen

insgesamt: 31 bytes

Und jetzt zur interessanten Frage: Warum und wie kann das einen Buffer Overflow auslösen?

Die Antwort: Mit nur einer Zeile im Log (also nur ein Client, nur ein Browsertyp) bekommt man 100%. Das sind 3 Ziffern, die dann den String ein Byte länger machen. Das gibt dann insgesamt 32 Byte und der Platz reicht nicht mehr fürs Nullbyte am Ende

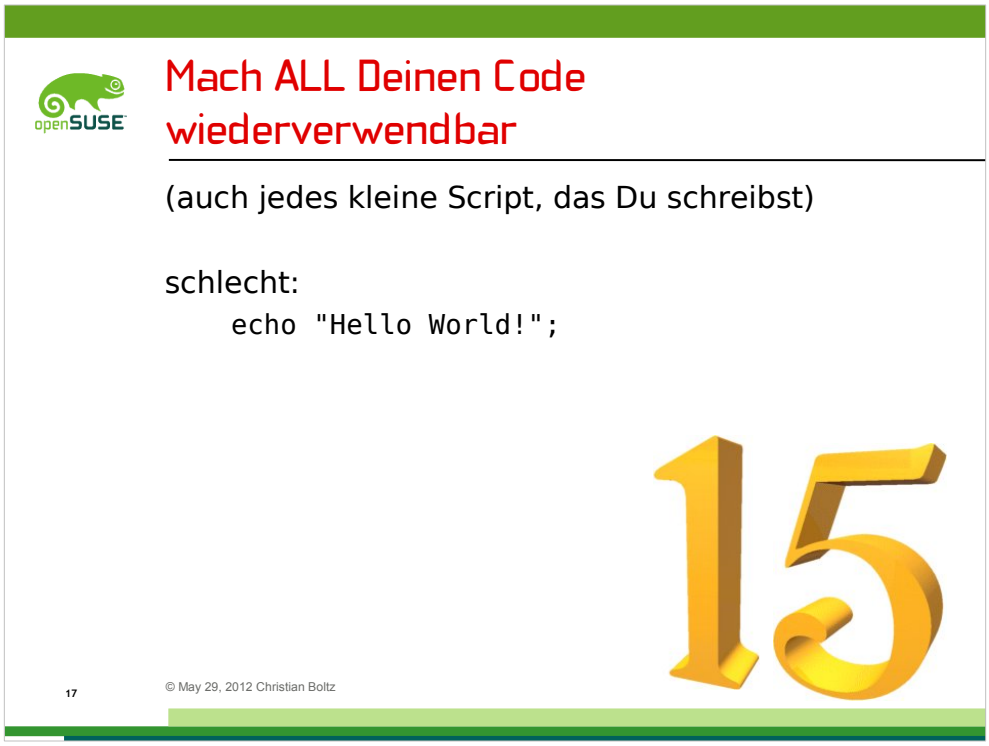
Mach Deinen Code niemals wiederverwendbar

… vor allem wenn er mehr als 1000 Zeilen hat

Code wiederverwenden ist wie Sätze aus Texten anderer Leute zu nehmen und zu versuchen, einen Magazin-Artikel daraus zu machen.
[Bob Frankston, übersetzt]

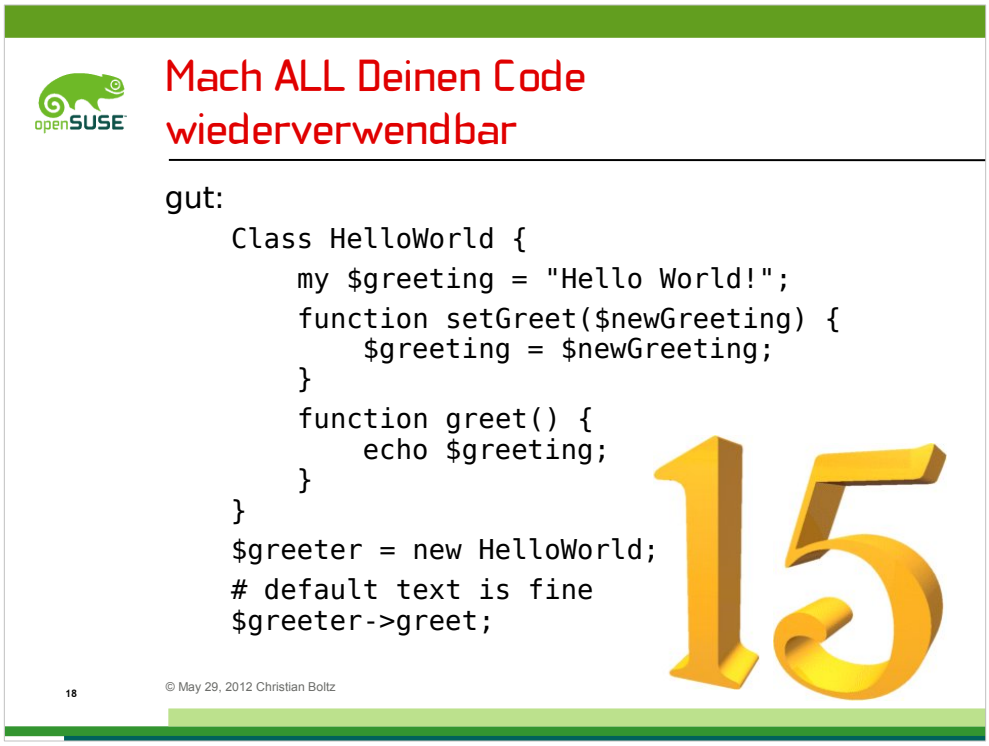
Davon abgesehen:

Niemand braucht soviel Code ein zweites Mal ;-)



Das ist schlechter Code – er ist nicht wiederverwertbar.

Machen wir ihn besser...

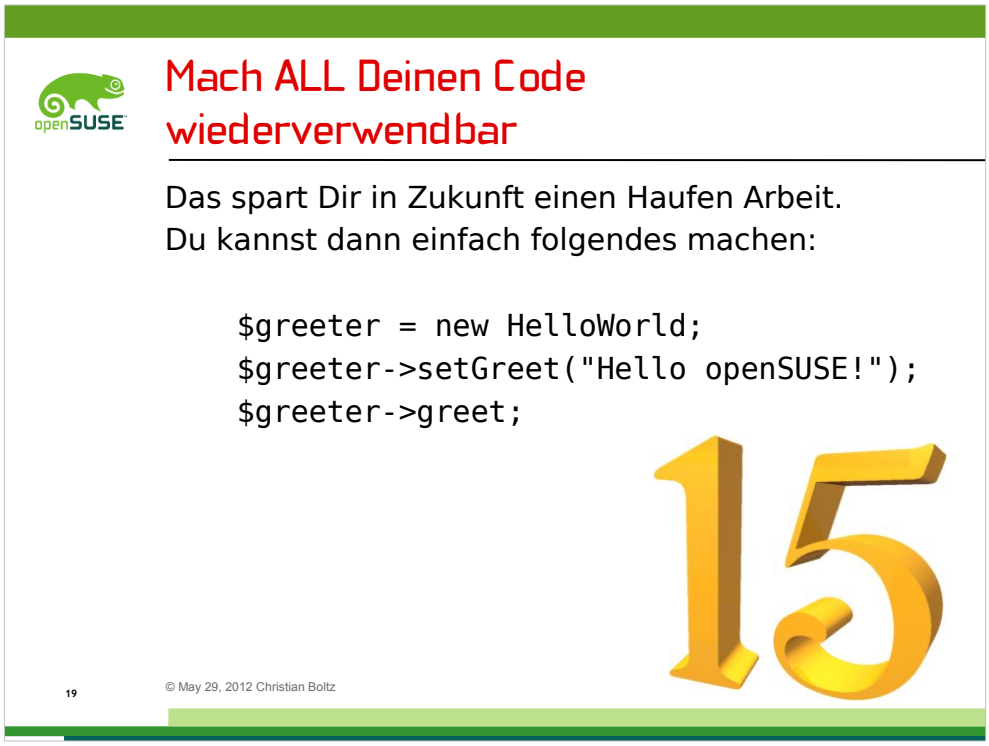


Hier ist der gute Code:

- der Grußtext ist in einer Variable – der Code ist also flexibel genug, um mit verschiedenen Texten zu funktionieren

- alles ist in einer Klasse gekapselt – die Internas sind nicht nach außen sichtbar

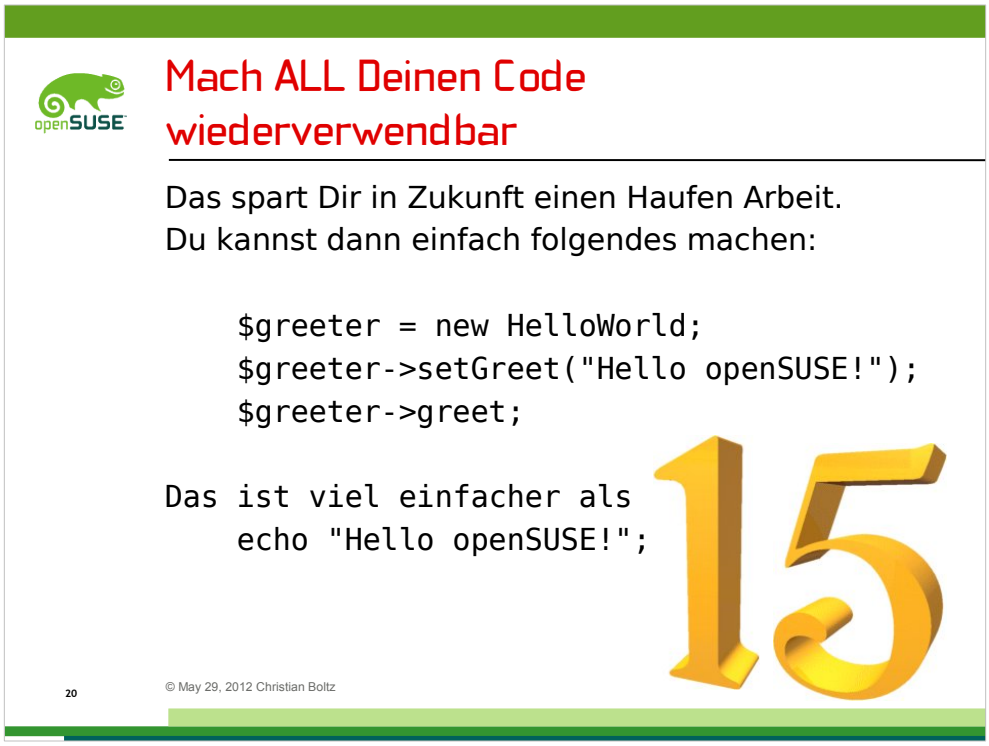
- und das Wichtigste: es ist wiederverwertbar



Seht Ihr, wie viel Arbeit wiederverwertbarer Code in Zukunft sparen kann?

Erstelle einfach eine Instanz der Klasse, lege den Grußtext fest und lass dann den Grußtext ausgeben

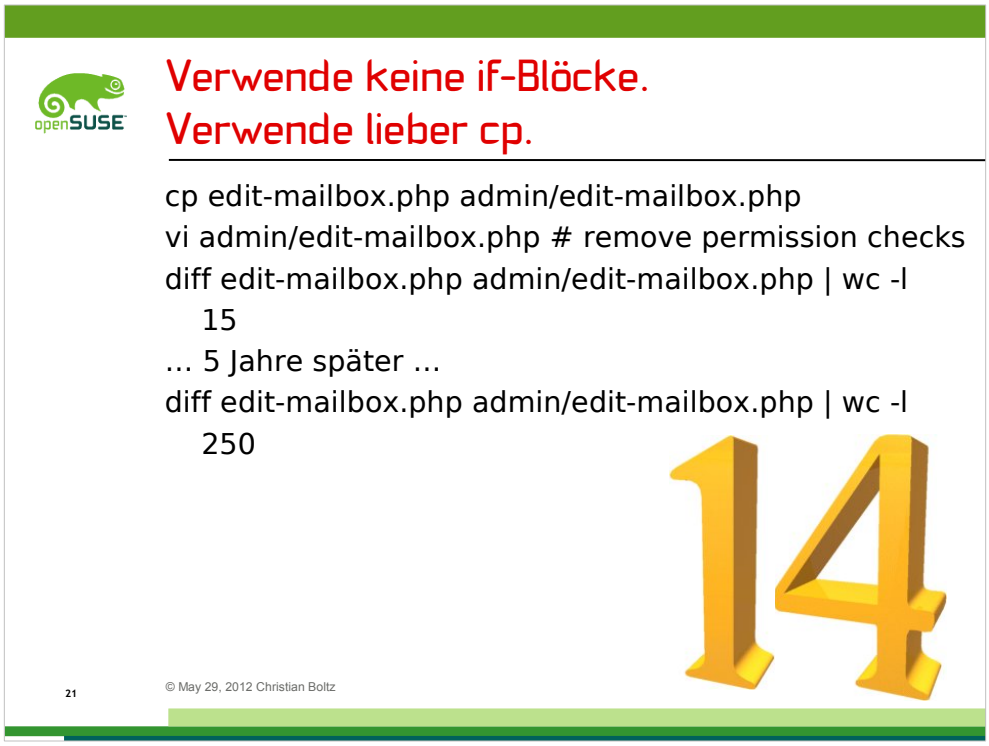
Und, am wichtigsten...



… das ist viel einfacher als Code zu verwenden, der nicht wiederverwertbar ist ;-)

In der Praxis liegt die Wahrheit irgendwo zwischen dieser und der vorherigen Regel – große Programme sollten Einheiten mit wiederverwertbarem Code haben, wo es Sinn macht. Andererseits macht es keinen Sinn, alles wiederverwertbar zu machen, wie dieses Hello World-Beispiel ja gezeigt hat.

Normalerweise ist der pragmatische Ansatz der Beste – mach was Du für die beste Lösung hältst



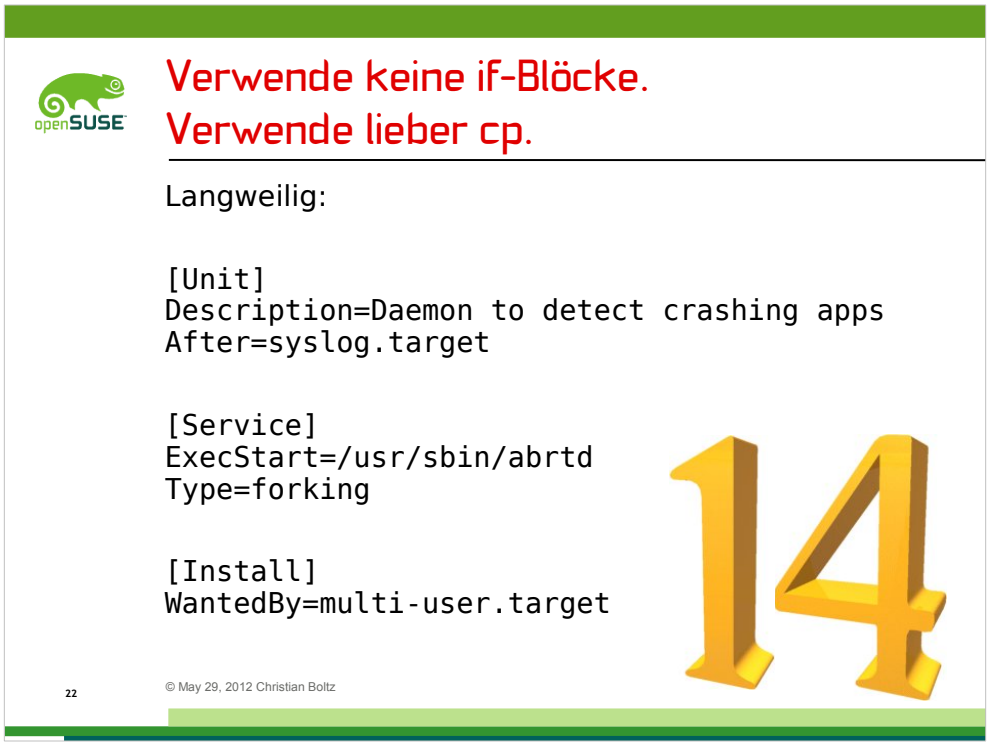
Das ist ein echtes Beispiel aus PostfixAdmin, nur die Anzahl der Jahre ist geschätzt.

Vor einigen Jahren hatten edit-mailbox (für Admins, die nur Rechte für ein paar Domains haben) und admin/edit-mailbox (für Superadmins, sowas wie root) den gleiche Code, mit der Ausnahme, dass für Superadmins die Domainrechte nicht überprüft wurden.

Über die Jahre gab es dann diverse Änderungen und Bugfixes – allerdings nur in einer Kopie des Codes.

Das Ergebnis waren mehrere Kopien des nahezu gleichen Codes, aber jede Kopie hatte einen eigenen Satz an Bugs.

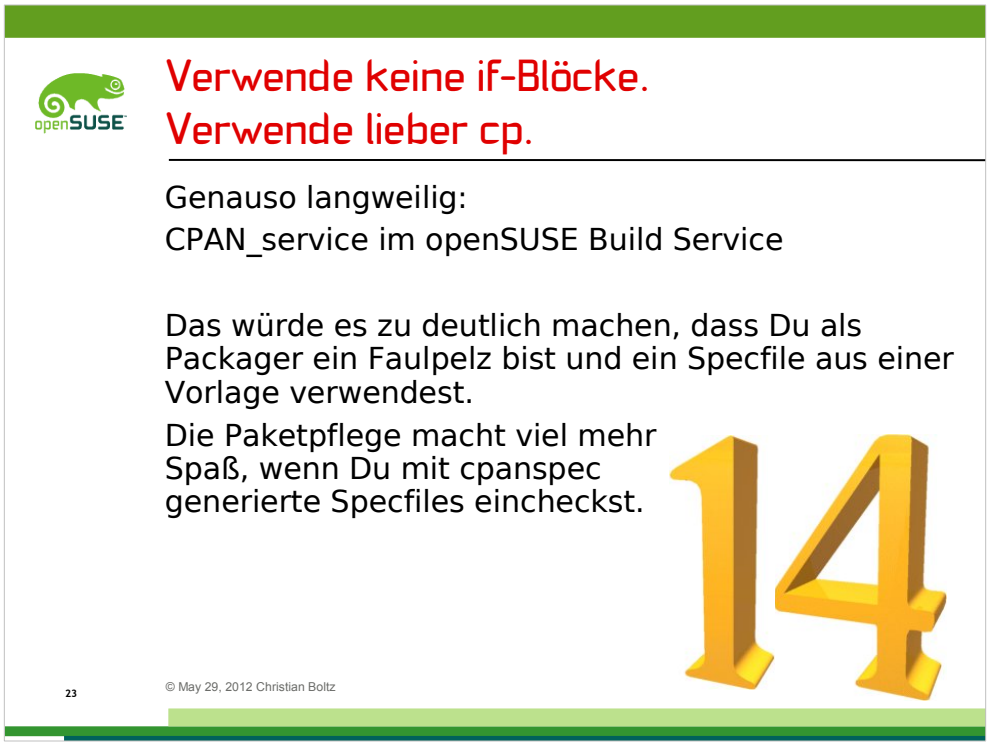
Genau das fand ich vor, als ich 2007 begann, an PostfixAdmin zu arbeiten. Seitdem habe ich einen Haufen Code Cleanup betrieben, doppelten Code entfernt und einen Großteil in Klassen umgewandelt. Immerhin haben wir auch ein paar coole neue Features eingebaut.



Ein Beispiel, das wohl jeder kennt, sind Initscripte.

Ihr wisst alle wie lang die sysvinit-Initscripte sind. Und wenn man sie mal vergleicht, stellt man fest, dass 80 oder 90% des Codes in allen Initscripten gleich ist.

Jetzt vergleicht das mal mit den systemd Unit Files. Die sind definitiv zu kurz, langweilig und viel zu pflegeleicht.



Eher spezifisch für openSUSE ist der Build Service und seine Source Services. Die sind im Allgemeinen eine gute Idee (oder wären es, wenn sie funktionieren würden)

Zum Beispiel gibt es einen cpanspec-Service, der 90% der Perl-Pakete automatisiert handhaben könnte.

Das würde aber zu deutlich machen, dass der Packager ein Faulpelz ist und ein Specfile aus einer Vorlage verwendet.

Die Paketpflege macht viel mehr Spaß, wenn man per cpanspec generierte Specfiles eincheckt. Wenn man die irgendwann updated, hat man die Wahl zwischen

a) die Versionsnummer im Spec ändern und hoffen, dass alles andere weiterhin funktioniert

b) cpanspec aufrufen, um ein neues Spec zu erstellen – und hoffen, dass man beim Aufruf von cpanspec keinen Zusatz-Parameter vergessen hat

Meine Lösung ist, dass ich ein “run-cpanspec.sh” in meine Pakete packe

Wenn es richtig lustig werden soll, diskutiert dsa Thema mal mit yaloki und darix – die hassen Source Services ;-)

Traue Deinen Benutzern
oder: überprüfe Benutzereingaben nie

Sogar AppArmor hielt sich an diese Regel, sie kann also nicht zu falsch sein ;-)

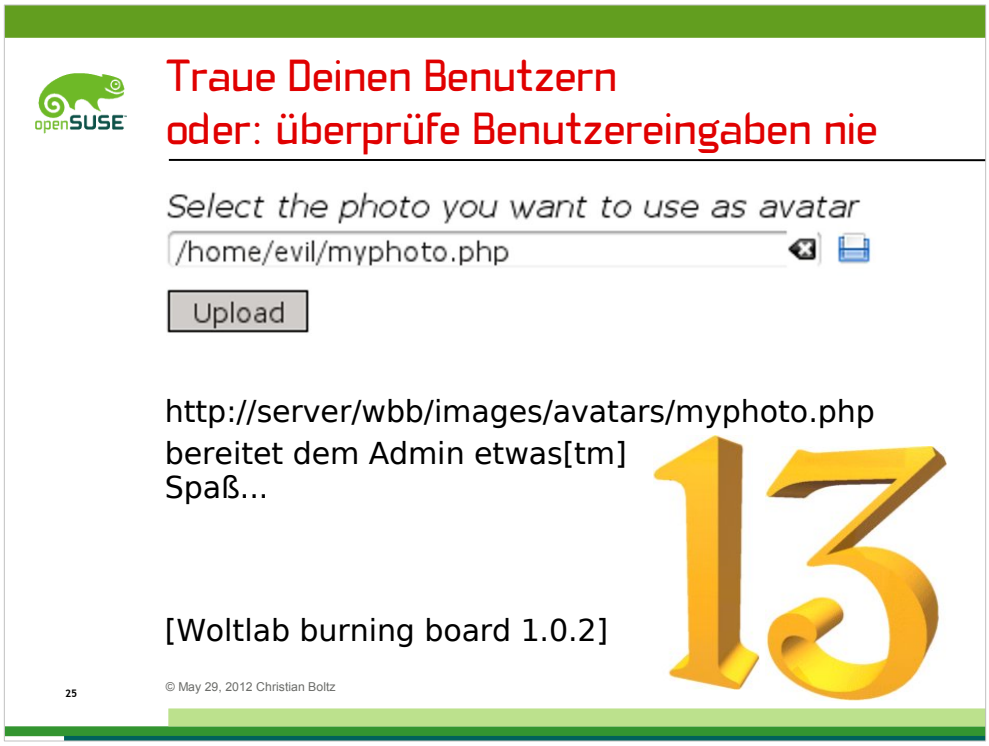
```
echo 'AAA AAA' > /proc/$$/attr/current
Segmentation fault
```

[107353.169142] kernel BUG at kernel BUG at /usr/src/packages/BUILD/kernel-desktop-2.6.37.6/linux-2.6.37/security/apparmor/audit.c:183!

[107353.169159] invalid opcode: 0000 [#7] SMP

[...]

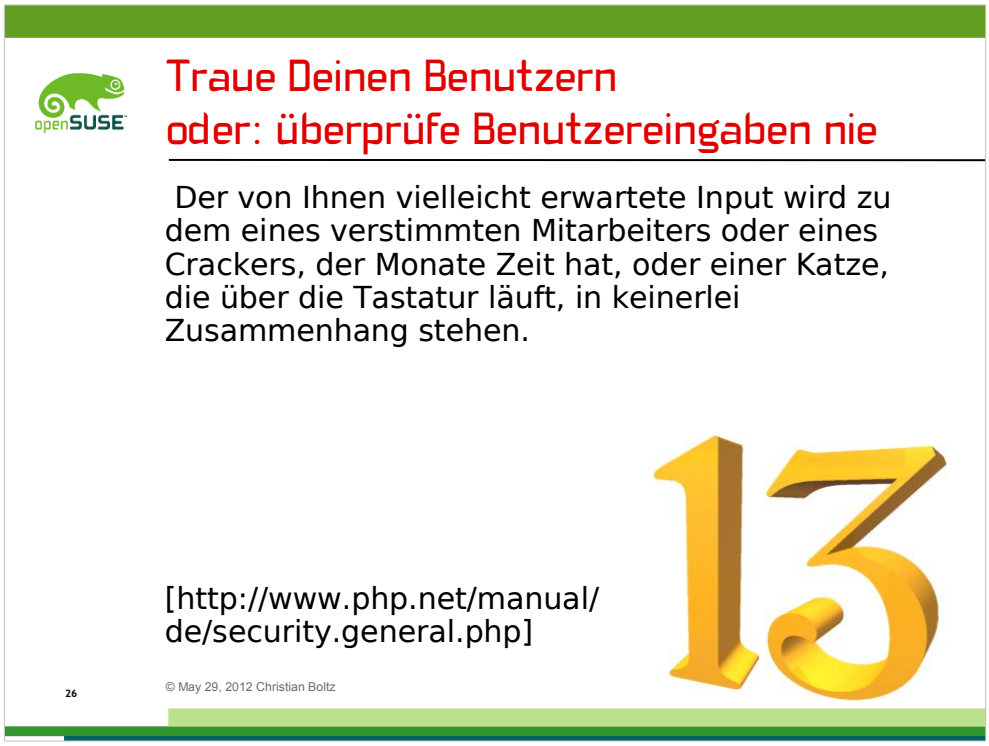
[https://bugs.launchpad.net/bugs/789409]



Real passiertes Beispiel: Das Forum eines Kunden hat erlaubt, beliebige Dateien als Avatarbild hochzuladen. Einschließlich PHP-Scripten...

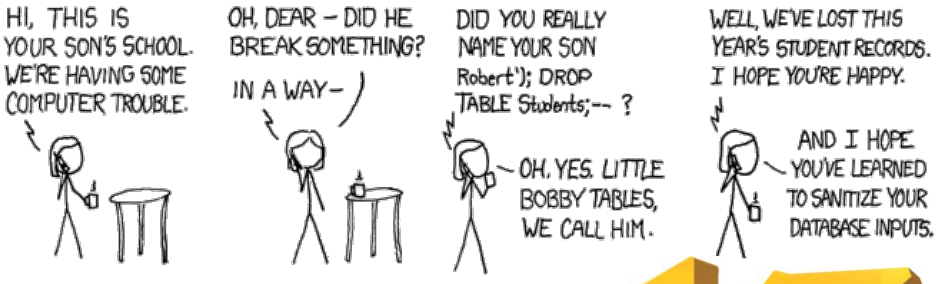
Nebenbei: die nächste Version hat das dann behoben, aber im Gegenzug erlaubt, beliebige Dateien runterzuladen, soweit sie der Webserver lesen kann. Sehr beliebt sind immer /etc/passwd und PHP-Dateien mit Datenbank-Zasswördern.

Ich würde sagen, das PHP Security Team liegt nicht ganz falsch. Die sagen...

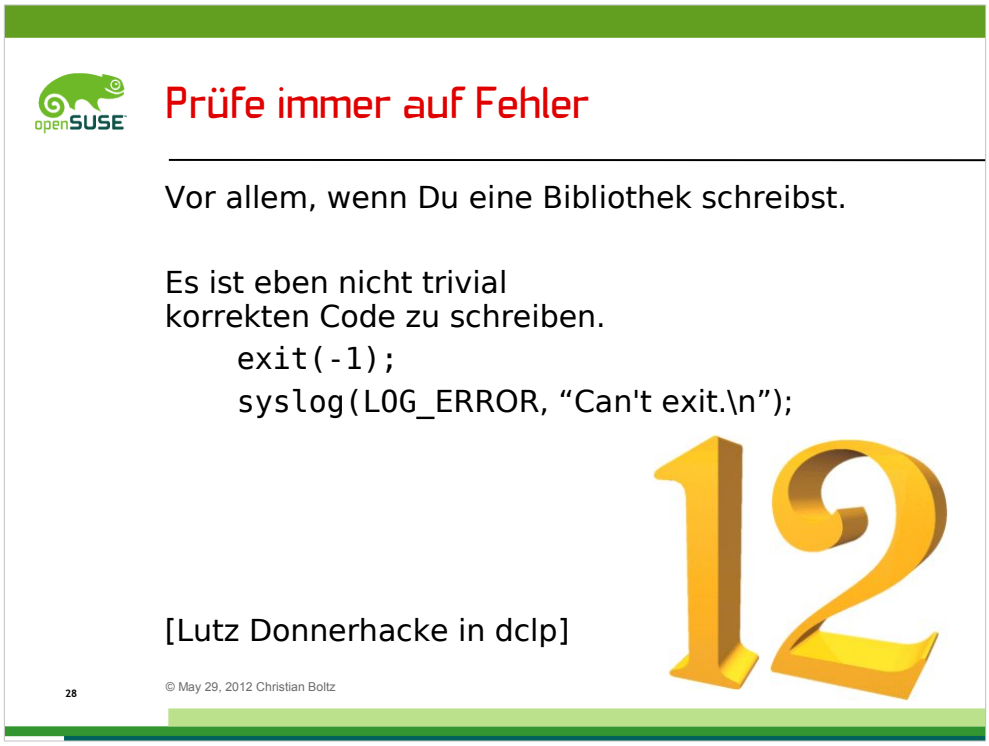


… aber diese Regel würe nicht komplett ohne Little Bobby Tables...

Traue Deinen Benutzern
oder: überprüfe Benutzereingaben nie



[http://xkcd.com/327/]



Diese Regel gilt vor allem, wenn man eine Bibliothek schreibt.

Beispiel: eine Memory Allocation schlägt mangels freiem Speicher fehl. Was tun?

Gute Frage ;-) Es gibt Fälle, in denen man nicht weiß, wie man einen Fehler behandeln soll. Dann ist es besser, den Fehler an die aufrufende Applikation weiterzureichen, als irgendwas zu machen, das die Applikation zerschießt. In diesem Fall bitte alle Details des Fehlers weiterleiten – die Info “irgendwas ist schiefgegangen” ist wenig hilfreich.

Zurück zum Speichermangel:

Die Bibliothek kann nicht einfach eine Fehlermeldung ausgeben und “exit” aufrufen – damit würde auch das aufrufende Programm an einer undefinierten Stelle abgebrochen.

Die einzige Lösung, die ich akzeptieren würde, ist: die Bibliothek soll schnell im Internet kostenlose Speichermodule bestellen ;-)

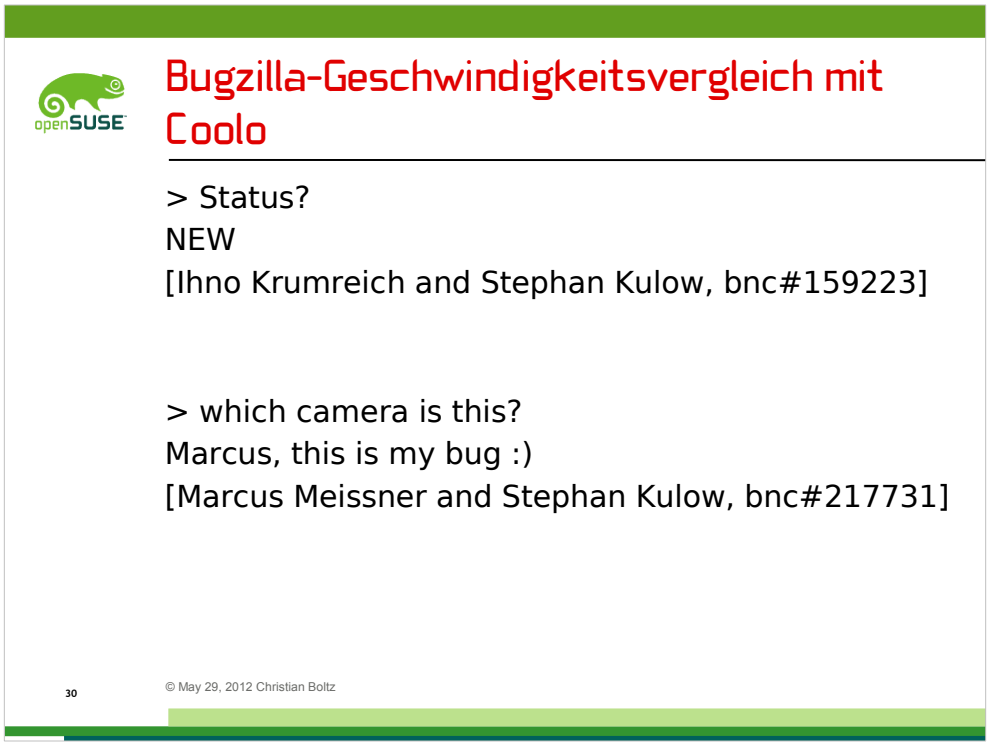
Oder, eher ernsthaft, dem aufrufenden Programm die Fehlermeldung “Speichermangel” weiterreichen und hoffen, dass die Applikation das vernünftig handhabt.

Die Schlussfolgerung ist also...

Denke nie an Fehlerbehandlung

Erwarte einfach, dass Dein Code funktioniert.

- Fehlerbehandlung ist nur langweilige Programmierarbeit. Es ist viel aufregender, Fehler nach dem Deployen auf wichtigen Systemen zu fixen
- Investiere Deine Zeit lieber in die Entwicklung neuer Features
- davon abgesehen ist das Nachdenken über Fehler sowieso die Aufgabe von Bugreportern



Bevor wir zu den Top 10 kommen, noch ein kleiner Bugzilla-Geschwindigkeitsvergleich mit Coolo.

Der erste Bug war ewig offen, daher die Nachfrage nach dem Status, der mit “NEW” technisch völlig korrekt beantwortet wurde.

Der zweite Bug war begehrter – den ließ sich Coolo nicht von Marcus wegschnappen ;-)

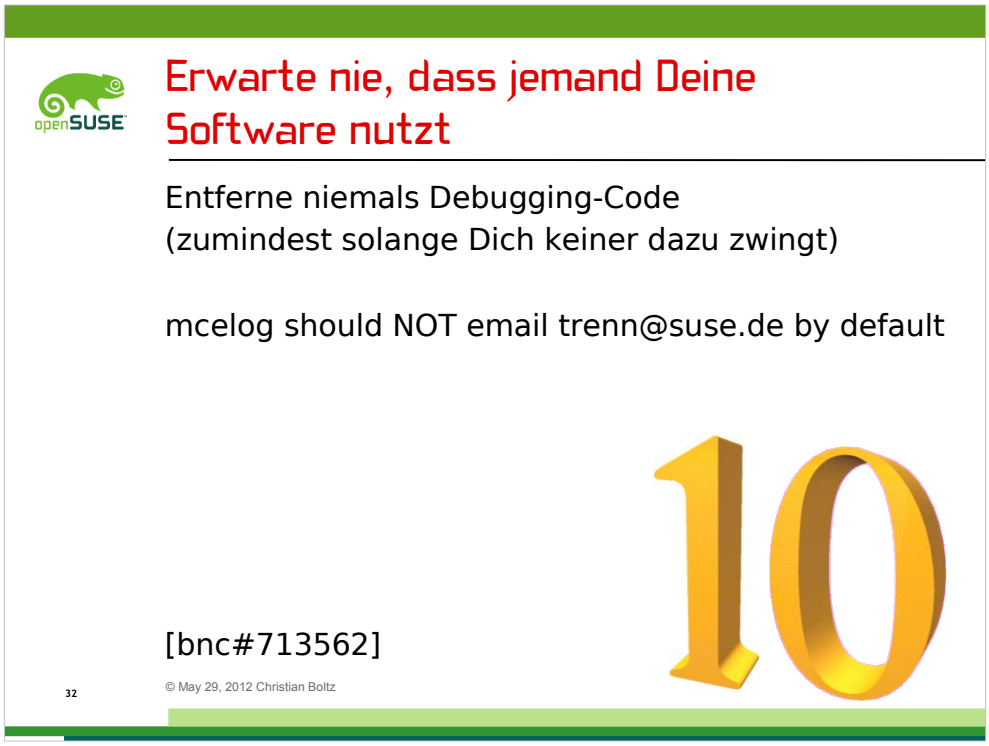
Erwarte nie, dass jemand Deine Software nutzt

Schreib ruhig Deinen Benutzernamen hartcodiert rein.

> mkdir: Can't create directory »»/home/ratti««
> Du solltest lieber ~ statt
> /home/ratti verwenden ;-)

Warte nur ab, die 0.0.3 funktioniert sogar, wenn man nicht "ratti" heisst. ;-)

[Spaß mit Ratti bei der Fontlinge-Entwicklung]



Der ein oder andere hat vielleicht das Update für openSUSE 11.4 gesehen, das vor ein paar Monaten genau dieses Problem behob.

...

und schließlich das bekannteste Beispiel von jemand, der nie erwartet hätte, dass seine Software von irgendjemand verwendet wird...

Erwarte nie, dass jemand Deine Software nutzt

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

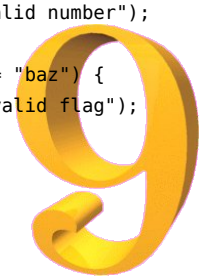
[Linus Torwalds, August 1991]

Verschachtle Deinen Code so tief wie möglich

```
if ($name == “foo”) {
    if ($value == “bar”) {
        if ($number > 0) {
            if ($flags == "baz") {
                # 1000 Zeilen Code
            } else {
                die ("invalid flag");
            }
        } else {
            die ("invalid number");
        }
    } else {
        die ("invalid value");
    }
} else {
    die ("invalid name");
}
```

Mach niemals sowas:

```
if ($name != “foo”) {
    die("invalid name");
}
if ($value != "bar") {
    die ("invalid value");
}
if ($number <= 0) {
    die("invalid number");
}
if ($flags != "baz") {
    die ("invalid flag");
}
# 1000
# Zeilen
# Code
```

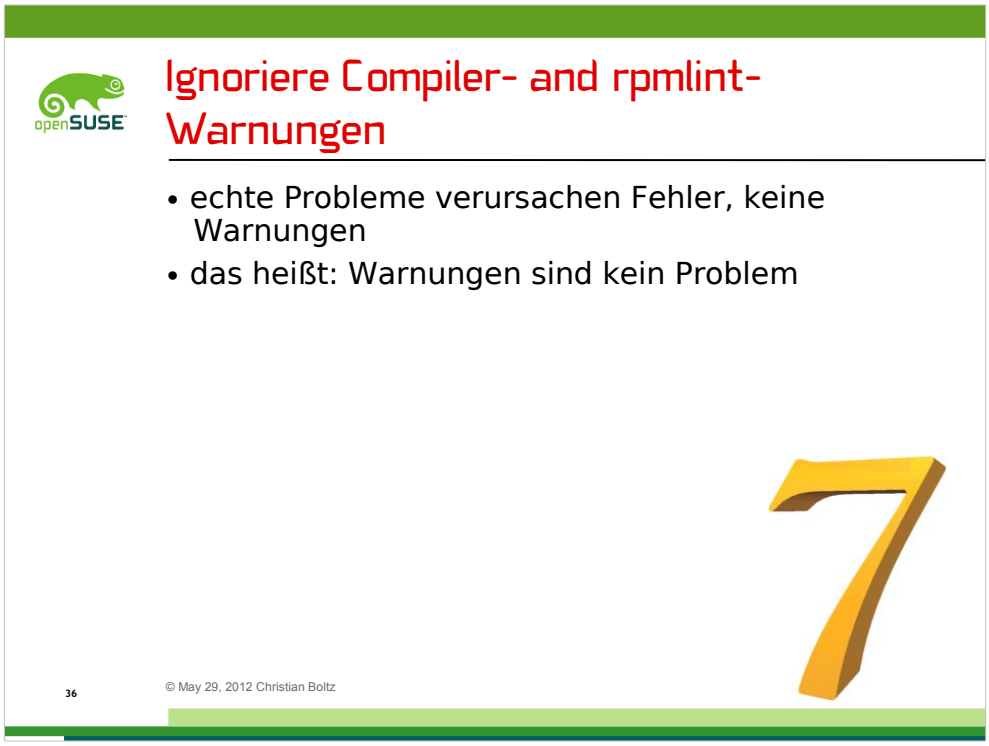


Mach Gedächtnistraining mit Deinen Benutzern

```
# zypper ar --help |grep refresh
-f, --refresh     Enable autorefresh of the repository.
# zypper mr --help |grep refresh
-r, --refresh     Enable auto-refresh of the repository.
```

[bnc#661410]

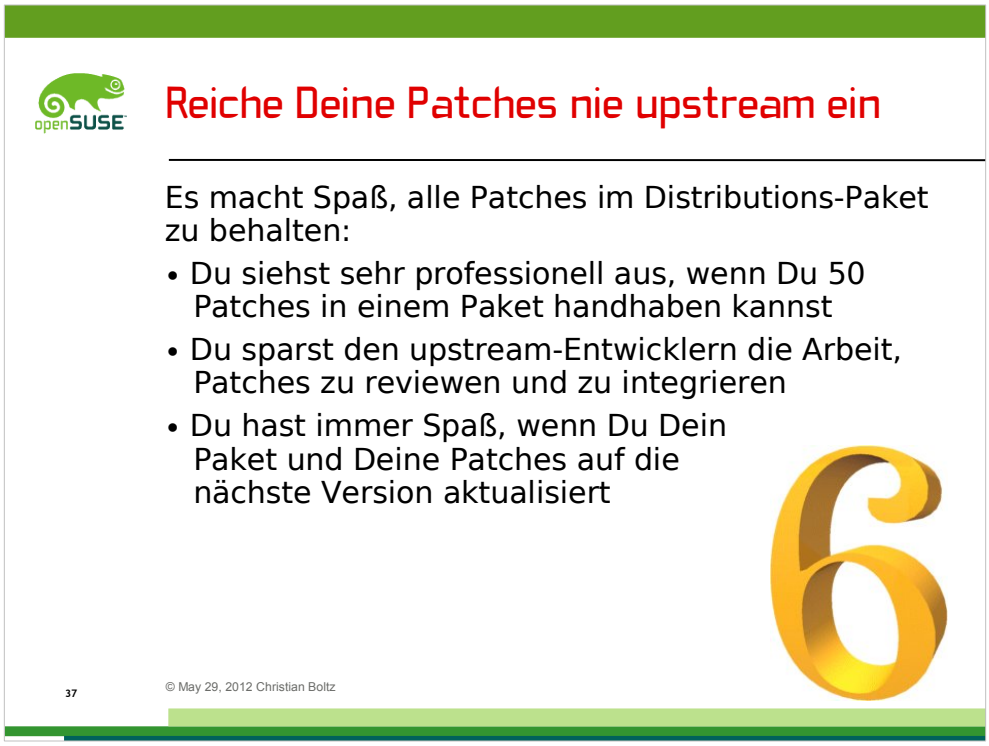
Es wäre doch langweilig, wenn dieselbe Kurz-Option für beide funktionieren würde, oder?



Beispiele:

- nicht initialisierte Variablen kann es nicht geben. Der Speicher ist immer voll, also müssen diese Variablen irgendeinen Inhalt haben

- und es kommt garantiert niemals vor, dass man sich bei den Variablennamen vertippt



Diese Folie ist dem openSUSE AppArmor-Paket gewidmet, passt aber wahrscheinlich noch auf mehr Pakete.

Ich würde sagen, dass das AppArmor-Paket mit 24 Patches ziemlich professionell ausgesehen hat.

Und ich kann bestätigen, dass das Upstreaming der Patches den AppArmor-Entwicklern einige Review-Arbeit gemacht hat. Ich habe die meisten Patches upstream gepostet, und es hat einige Tage gedauert, biss die Entwickler sich von meiner Patch-Flut erholt hatten.

Auf der openSUSE conference im September hatte ich noch erwähnt, dass derjenige, der das AppArmor-Paket auf 2.7 aktualisiert, weniger als 10 Patches übrighat. Inzwischen bin ich der Glückliche – und habe erstmal einen 370 kB Monsterpatch gekillt, der die AppArmor-Makefiles durch automake ersetzt hat, sich auf den 2.7er Code nicht mehr anwenden ließ und daher upstream wohl nicht angenommen worden wäre.

Es gibt übrigens noch einen Grund, Patches nicht upstream zu schicken: Das openSUSE-Paket sollte immer besser als die Pakete in anderen Distributionen sein. Daher ist es immer gut, ein paar wichtige Patches nicht weiterzugeben.

Mache ich übrigens auch – seit openSUSE 12.1 wird das AppArmor-Profil für Samba automatisch um die Rechte für alle Shares ergänzt.

Schreibe niemals Dokumentation

- falls schon alte Doku existiert, aktualisiere sie auf keinen Fall
- Doku liest eh keiner
- Kommentare im Code zählen auch als Doku - vermeide sie nach Möglichkeit

Schreibe niemals Dokumentation

- Doku liest eh keiner

Wirklich?

Ich habe die 10.1 wie ein echter User getestet. In anderen Worten: Ich habe nie die Release Notes oder die Doku gelesen ;-)

[tomhorsley(at)adelphia.net
in opensuse-factory, übersetzt]

Die meisten User lesen keine Doku.

Aber es kommt durchaus vor, dass der BBfH mal wieder zuschlägt

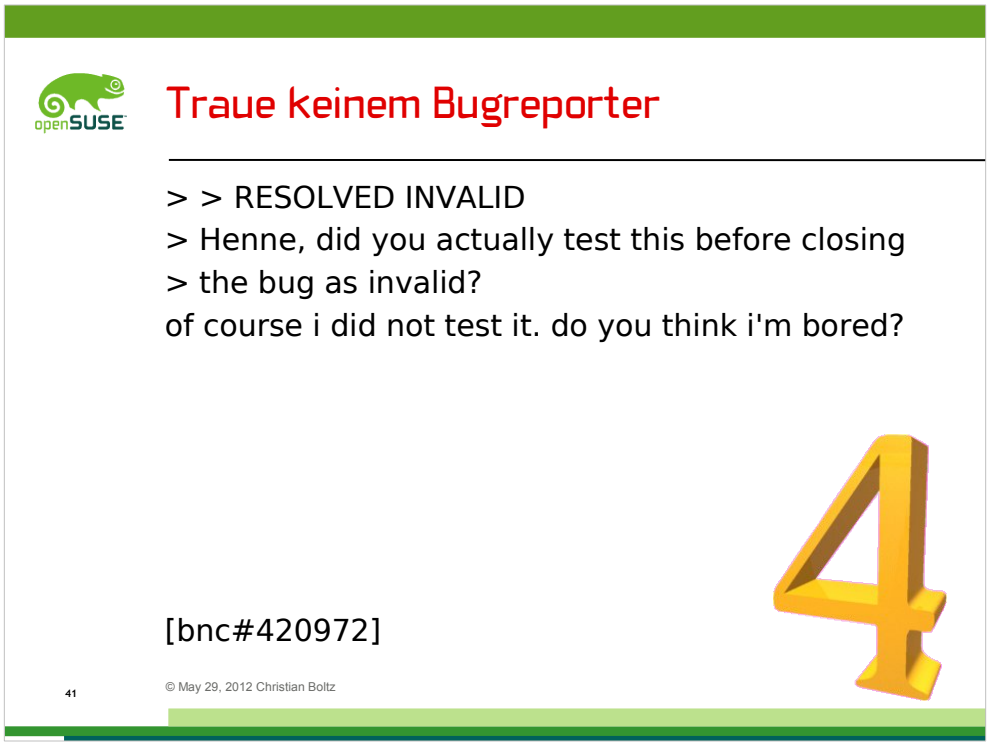
Erinnert sich noch jemand an die gute alte Zeit, als es noch gedruckte Handbücher zu damals noch SUSE Linux gab? Hier ist eins davon.

Jede Büroklammer markiert einen Fehler im Kapitel zur Shell.

Das Ergebnis war ein Bugreport über 3 DIN A4-Seiten.

Erste Rückmeldung in Bugzilla:

@bugreporter: please do not ever touch a paperclip again ... :)



Ihr erinnert Euch? Bugreporter sind böse

Sie klauen Deine Zeit, indem sie Fehler melden und erwarten, dass sie gefixt werden. Natürlich von Dir, dem mehr oder weniger glücklichen Empfänger des Bugreports.

Und, noch schlimmer, wenn man versucht, einen Bug als invalid zu schließen, um ihn loszuwerden, klicken sie einfach auf reopen...

In diesem Fall war es übrigens eine Regression im courier-imap imapscript.

Unnötig zu erwähnen, dass der Bugreport berechtigt war...

Traue keinem Bugreporter

general rule: if Olaf reports a bug, it is a valid bug.
(Olaf Hering while reopening bnc#168595)

Spaß mit NEEDINFO

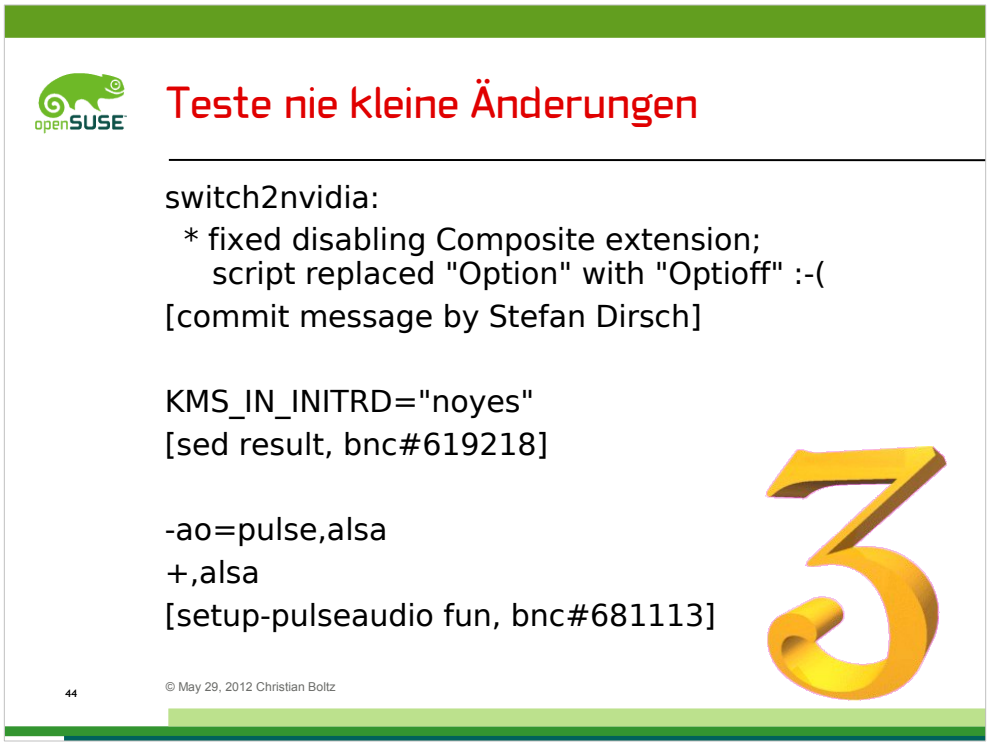
I am supposed to be the info provider,
so here is my answer:

42

By the way:

What is the question?

[Johannes Meixner, bnc#190173]



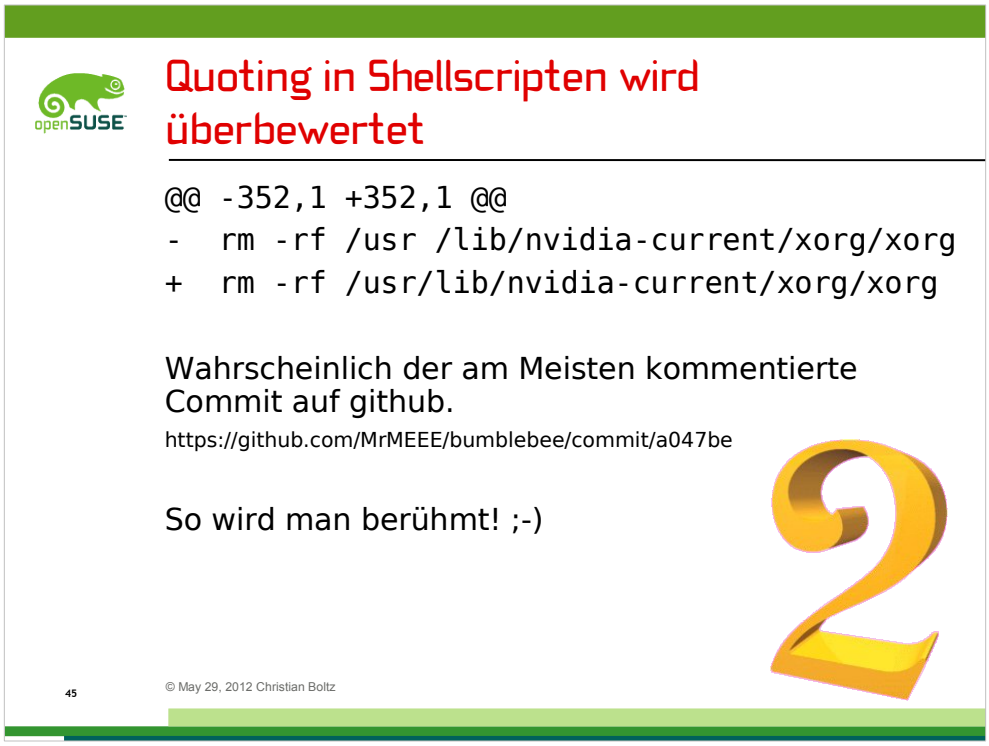
gültig für Änderungen bis 200 Zeilen ;-)

Man findet vermutlich viele ähnliche Beispiele. Hier ist eine Auswahl der interessantesten:

Zuerst das “optioff” - hat übrigens funktioniert und erfolgreich die composite extension deaktiviert

Im zweiten Beispiel konnte sed sich nicht entscheiden und hat dann ein klares Jein, also “noyes” eingefügt, damit alle glücklich sind.

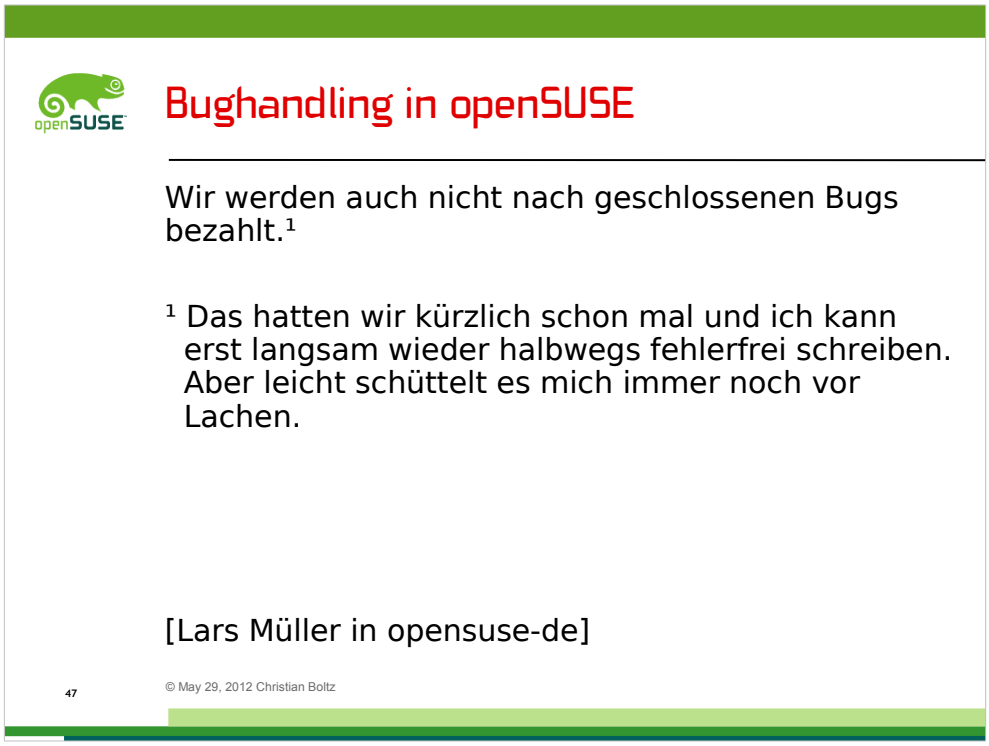
Und schließlich setup-pulseaudio, das mal eben die mplayer-Konfiguration zerschossen hat. Zu allem Überfluss hat mplayer in seiner Fehlermeldung nicht gesagt, in welcher Datei der Syntaxfehler ist. Sprich: man hat länger Spaß beim Suchen des Fehlers...



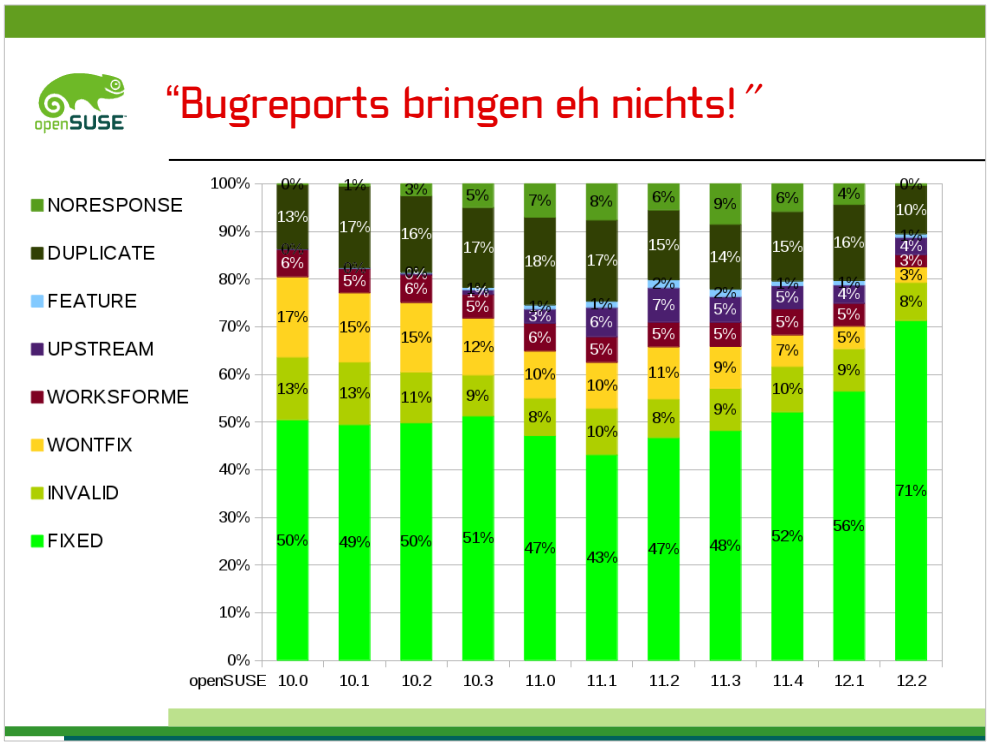
…

Bevor wir zu Regel Nummer eins kommen, werfe ich noch einen kurzen Blick auf die Bugzilla-Statistik.

Zum Bughandling in openSUSE gibt es verschiedene Gerüchte und Theorien. Lars Müller, ein SUSE-Mitarbeiter, hat mal ein verbreitetes Gerücht ausgeräumt.



Zum Bughandling in openSUSE gibt es verschiedene Gerüchte und Theorien. Lars Müller, ein SUSE-Mitarbeiter, hat mal ein verbreitetes Gerücht ausgeräumt.



Den Spruch "Bugreports bringen eh nichts!" hat man schon mehr als einmal gehört. Daher ein kurzer Blick in die Bugzilla-Statistik der geschlossenen Bugs verschiedener openSUSE-Versionen.

Rund 15% sind Duplikate – das Risiko, einen bereits bekannten Fehler zu berichten, hält sich also in Grenzen – oder die meisten Bugreporter wissen, wie man die Suche bedient.

Die nächsten 15% sind INVALID oder WORKSFORME, also z. B. Bedien- oder Konfigurationsfehler, die keine Bugs sind oder nicht reproduzierbare Probleme.

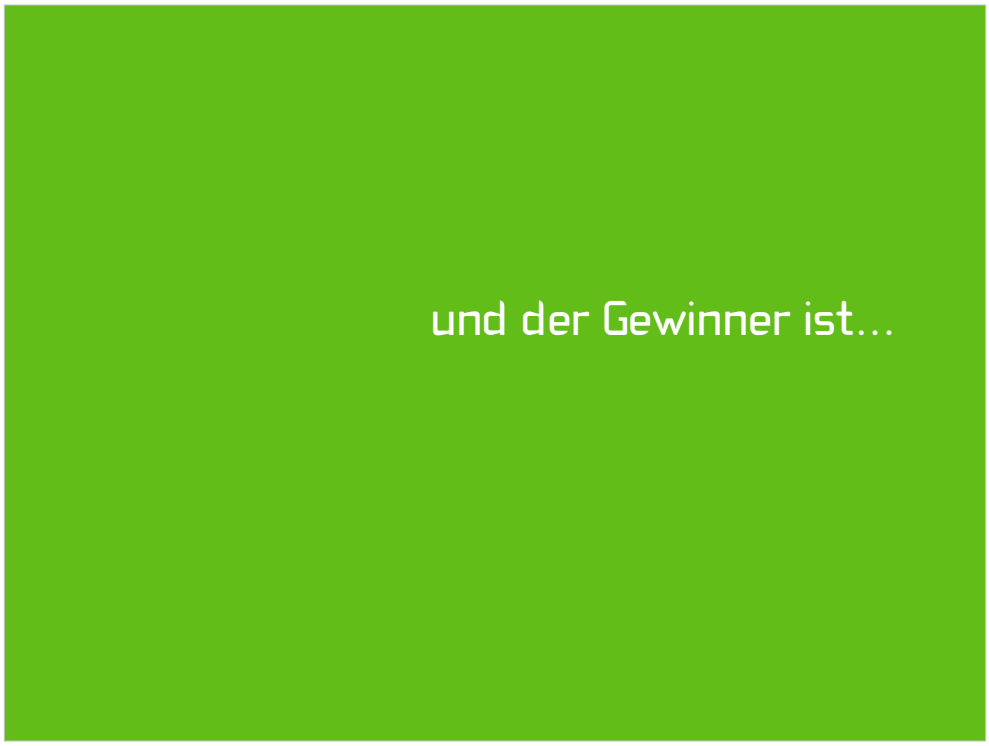
5% der Bugreports versanden mangels Rückmeldung der User

Weitere 15% verteilen sich auf WONTFIX, UPSTREAM und FEATURE

Die wichtigste Erkenntnis: über 50% aller Bugreports wird tatsächlich gefixt – auch wenn es manchmal "etwas" länger dauert. Bei 4000 Bugreports pro Release sind das 2000 gefixte Bugs.

Daher nochmal eine Aussage von Lars Müller: "Bugzilla beißt nicht und ist viel, viel netter als ich ;)"

[[derzeit ca. 4000 Bugs offen]]



es wird spannend...

Meine Damen und Herren!

Hier ist die Nummer Eins der goldenen Regeln für schlechte Programmierung!

*** Trommelwirbel ***

Dieser Code spricht wohl für sich.

Ich spendiere ein openSUSE Bier (??????) für den ersten, der mir in den nächsten 5 Minuten sagt, was dieses kleine Programm macht. Ja, es ist gültiger Perl Code.

.

Ich zeige dieses Programm gleich nochmal. Vorher noch eins:

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

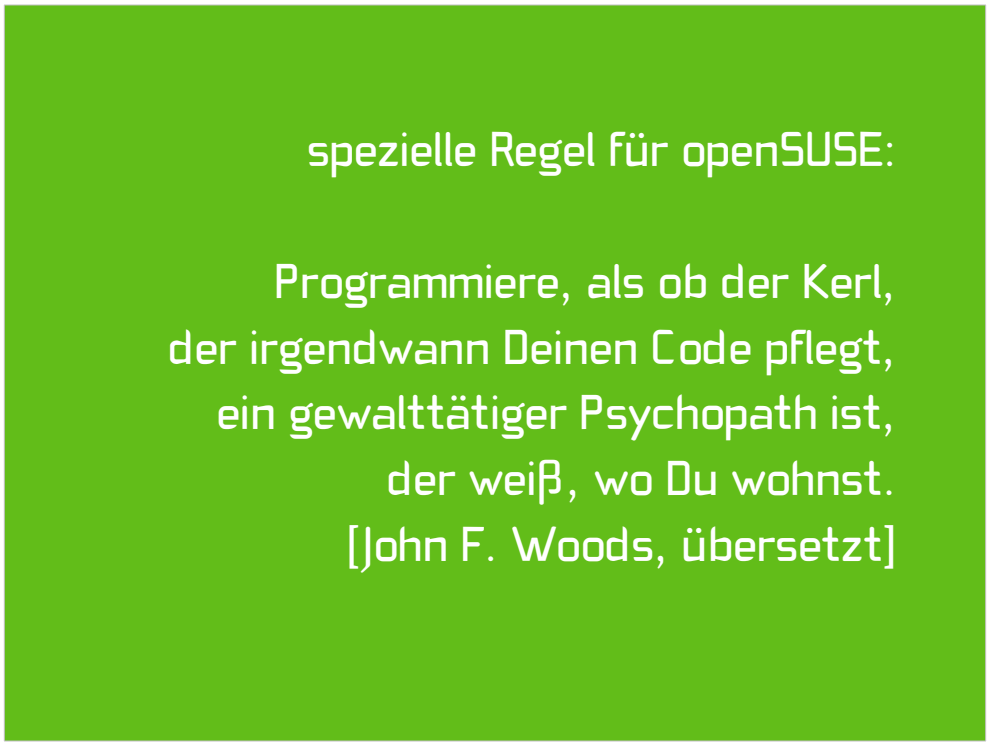
.

.

.

.

Es gibt eine spezielle Regel für openSUSE, und sie ist vielleicht wichtiger als alle anderen Regeln, die ich gerade erklärt habe:



Es gibt eine spezielle Regel für openSUSE, und sie ist vielleicht wichtiger als alle anderen Regeln, die ich gerade erklärt habe:

Danke!

- jedem, der versehentlich ;-) zu meinem Vortrag beigetragen hat
- für die Inspiration bei
  http://www.karzauninkat.com/Goldhtml/
  “Goldene Regeln für schlechtes HTML”
  http://www.sapdesignguild.org/community/design/golden_rules.asp
  “golden rules for bad user interfaces”
  http://blog.koehntopp.de/archives/2127-The-Importance-Of-FAIL.html
  http://blog.koehntopp.de/archives/2611-Was-bedeutet-eigentlich-Never-check-for-an-error-condition-you-dont-know-how-to-handle.html
  zwei großartige Artikel über FAIL von Kris Köhntopp
- perl Acme::EyeDrops fürs Rendern von Regel #1
- fürs Zuhören

Fragen?
Anmerkungen?
Flames?