Tests and bugreports

@ItSANgo

testとbugreportについて話をしたいと思います。

Who am I

- @ItSANgo
- http://d.hanena.ne.jp/Itisango/
- http://mixi.jp/show_profile.pl?id=789759
- http://www.facebook.com/profile.php?id=100005833863069
- 無職透明
  - 時間だけはたっぷりある
- TOEIC 355点
  - 英語は話せない

Netでは135を名乗っています。
現在、職についていません。暇だけが資本です。
有意性は切れていますがTOEIC355点ということで英語は全然できません。

何を話すか

- Tests
- Bugreports

- 間違っていることが書いてあったら指摘してください(_ _)
  - 今ここで
  - TwitterやBLOG等にコメントでも

3

Softwareのtestとbugreportについてお話ししようと思います。

時間の都合で全てお話しすることができないかもしれません。

この資料はnetからdownload出来るようにすることを予定しています。
お話できなかったところはdownloadした資料を参照してください。
入手方法は東海道らぐmailing list等でアナウンスする予定です。

間違っているところも多々あると思いますが、都度、ご指摘お願いいたします。
公開後はTwitterやBLOG等で誤りを指摘していただくと助かります。

Tests

まずはtestについて話をしたいと思います。

Testといってもいろいろあるよね

- http://ja.wikipedia.org/wiki/%E3%82%BD%E3%83%95%E3%83%88%E3%82%A6%E%82%A7%E3%82%A2%E3%83%86%E3%82%B9E3%83%88
- 単体test
  - sourceをdownloadしてmake checkを走らせるとか…
- 結合test
- ホワイトボックステスト
  - 無理(-_-)
- ブラックボックステスト

testといってもいろいろありますよね。
Wikipediaには様々なtestについて解説されています。
またtest関連の書籍などもいろいろ出ていると思います。

testのresource

- 人
  - tester
- 物
  - test対象
  - test環境
  - test仕様・test項目
- 金?
  - そんなものは無いw
- 時間
  - スケジュール
  - 納期

テストに必要なものをresourceとして挙げてみました。
test対象がないとtestにはなりません。
それにtesterやtest環境がいないとtestは始まらないですよね。
test対象があればtest仕様も当然あるはずですよね。
testには時間がかかるのでこれもresourceです。
多くのtestでは納期やスケジュールが有るはずです。

ここで扱うtest

- 対象
  - FLOSS
  - Linuxのdistribution
    - 網羅は不可能
  - リリース前test
    - RC版 、Alpha, Beta版
- Tester
  - 人数不定
  - 緩い統制
  - スキル不揃い
- Method
  - test仕様は?
- →題して「つれづれtest」

7

一般論は置いておいて、ここで扱うテストについて説明したいと思います。
ここでは主にopenSUSE等のLinuxのdistributionのリリース前testについて話をしようと思っています。
OSは大きいので網羅というのは不可能です。
OSのテストというのはsoftwareの規模に対してtesterの数は圧倒的に不足します。
testerもどこの組織に所属している訳でもない緩やかなcommunityのmemberであると考えられます。
testerのskillもまちまちで不揃いです。
testの方法論も確立しているのか、test仕様があるかどうかも怪しい状態です。
こんな中で「つれづれなるままに」testをしていくので、ここで扱うtestを私は「つれづれtest」と呼んでいます。

きっかけ

- opensuse-jaMLのmailから
  - http://lists.opensuse.org/opensuse-ja/2013-02/msg00002.html
- testと聞いたら血がたぎる

そもそもtestを始めたきっかけについて説明したいと思います。
最初は今年(2013年)2月にopensuse-ja mailing listに流れた一通のtest依頼mailがきっかけでした。
正直、私はtestが大好きです。testと聞いたら血がたぎります。
暇もあったのでopenSUSE 12.3 RC版を私は徹底的にtestすることにしました。

Testの方法は

- Test仕様がある場合
  - 例: Ubuntu-Japanese RemixのQAテスト
    - https://wiki.ubuntulinux.jp/Develop/Precise/QA/RemixCDImage
- Test仕様がない場合
  - 自分でtest仕様を作るしかない。
  - まじめに考えると結構重たい作業だが…

（quality assurance testing / 品質保証テスト）
Testの方法についてお話ししたいと思います。
testは仕様の有る無しによって大きく2種類に分けられると思います。
Ubuntu-Japanese RemixのQA testの様にtest仕様がある場合にはそれに従ってtestすれば良いと思います。
test仕様がない場合、自分でtest仕様を作る必要があります。真面目に考えるとものすごく手間のかかる重たい作業です。

openSUSE jaの場合

- 一応test項目は有る…けど緩やか
  - https://ja.opensuse.org/Test_Weeks

openSUSEの場合、一応test項目のワクはあったのですが、ざっくばらんで詳細が詰められていない感じでした。mailing listにも「作りかけ」とありました。

→「なら自由にやらせてもらおう」、というのが私の本音です。

test環境の準備

- VMが便利
  - VirtualBox個人的にお勧め
  - 特にinstallのtestで威力を発揮する。
- 実機でのtestも必要
  - 実機でしか起きないbug
  - すみません、あまりtestしていませんorz
- VMでしか起きないbugはどうする?
  - →報告する。
    - VMも動作環境だ!
    - VM上のbugが潜在的な問題を持っている可能性

testをするにはtest環境が欠かせません。
仮想環境があると重宝します。
installer周りのtestを繰り返す必要がある場合にはとても便利です。
個人的にはVirtualBoxがWindowsの上でも使えてしかも只なのでお勧めです。

実機と仮想環境は異なるので実機か仮想環境、どちらかでしか起きないbugもあり得ます。だから実機でのtestも必要です。が、私はinstall出来る実機を持っていなかったので実機でのtestはほとんどやっていません。Windows PCにinstaller DVDを突っ込んでbootするかどうかを確かめた程度です。

ところでVM上でしか起きないbugが出た場合、どうすべきだと思いますか?私はVMも動作環境だと考えています。またVM上の異常動作が実機にも影響する可能性もありますので、bugreportすることにしています。

test項目の洗い出し

- とにかくたくさん洗い出す
  - 網羅性
  - 掛け算で考える (ex: (実機 + VirtualBox + VMware) * (x86 + x64) * (DVD + netInstall) * (KDE + GNOME + Xfce + LXDE + xdm + server))
  - すべきこと
    - softwareの目的が達成されているか?
      - Ex: editor → 編集できるか?fileの保存はできるか?
  - やりたいこと
    - 自分の興味で自由にw
- test項目はmemoしよう。
  - plain textで充分w

test項目を洗い出します。
test項目は思いつく限りたくさん洗い出した方がいいと思います。
コツは掛け算というか、組み合わせで考えることです。例えば、実機と仮想環境で分けて考え、KDE・GNOME・Xfce・LXDEなどデスクトップ環境で分けて考え、32bit環境と64bit環境とで分けて考えるとか組み合わせで考えれば、あっという間に項目は膨らみます。

またtest対象のsoftwareの目的に応じて、すべきことを洗い出します。
例えばEditorなら編集できるか、fileの保存ができるか、などがtest項目になります。

あと、つれづれtestでは自分で個人的に試してみたいこともtest項目として書きだします。

test項目はplain textで良いので、どこかにmemoしておきます。

すべきこと

- testするんだからこれは必要だよねという項目を入れる
- test環境の準備もtest項目に入れてしまえ!
  - 私がtest項目に入れているのはこれ(openSUSEの場合)
    - chmod 1777 /var/crash
    - /etc/security/limits.confの編集
      - * soft core unlimited
    - /etc/sysctrl.d/60-core_pattern.conf
      - kernel.core_pattern =/var/crash/core.%e.%u.%t
    - Reboot
    - cat
    - C-¥ (/var/crash/ にcoreが吐かれるか?)
  - これらをcommandやeditorのtestを兼ねて実行する。

testすべきことにちょっとだけ補足します。
そのsoftwareが持っている基本的な機能は、たとえ、そんなところにbugは無いだろうという初歩的・基本的ことでも必ずtestします。
例えばLibreOffice BASEの場合、table・query・form・reportの作成を項目に入れます。
(ちなみにLibreOffice BASEのreportはエンバグしやすい項目なので必ずtestします。実際にFedora19でreportが作成できないというbugがありました。
https://bugzilla.redhat.com/show_bug.cgi?id=983809
)

ついでにtest環境の準備作業もtest項目に入れておきます。
例えば最近のLinux distributionはdefaultでcoreを吐かせないように設定されています。これをcoreを吐かせるように設定変更します。これがcommandやeditorのtestを兼ねているのでtest項目に入れます。

ちなみにここに書いている設定はどんなsoftwareのcoreも/var/crash/ directoryに格納するという設定です。openSUSEのtestの際にはとても便利です。

testの実行と見直し

- test結果をmemoする(OK/NG)
- NGの場合は何が起こったか記録
- testしていくうちに、新たにtestすべき項目が見つかる。
  - 漏れ
  - 怪しい動き→もっと条件を変えてtest
  - 探索型・発見型test(自分も仕様を知らないsoftのtest)
- それもtestに入れる
- test項目は膨らむ

実際にテストを実行します。
test成功・失敗をmemoしていきます。
testに失敗した時には何が起こったかをmemoします。
ところでtestしているうちにもっとtest項目を増やしたいと思うことがあります。
test項目漏れや、不思議な挙動をするのでもう少し調べたい時などがそれです。
そんなときは遠慮なくtest項目を増やしていきます。
softwareの仕様を調べながらtestしているときも次々とtest項目が思いつくと思います。
したがってtest項目はどんどん増えていきます。

トリアージ

- test項目が膨らみ自分の手に負えなくなったときに
- 掛け算を崩す
  - (実機 + VirtualBox + VMware) + (x86 + x64) + (DVD + netInstall) + (KDE + GNOME + Xfce + LXDE + xdm + server)
- 類似のtestを省く
- 自分が出くわしたら困ることを中心にテストする
- 過去の経験を基にtest項目を取捨選択する
- test項目自体は消さない
  - 何をtestしたか・していないかのevidence

でも、test項目が増えすぎたら手に負えなくなります。
そんなときにはtestを省略する必要があります。
test項目を書いたmemoには残しておいて、何をtestしているか・していないかを記録しておきます。
掛け算で考えていたところを崩して足し算で考え直します。
例えばslideの例では掛け算にすると(3*2*2*6=)72通りのpatternをtestしなければいけなくなりますが、足し算で考え直すことで13通りにまでpatternが減らせます。

bugっていたら困る機能を中心にtestします。
過去の経験からbugが出やすそうなところを残し、大丈夫だろうと思うところはtestを省きます。

トリアージが裏目に出ることもある

- こんなbugは出ないだろ→出ます
  - https://bugzilla.novell.com/show_bug.cgi?id=829298
- 64bit環境と32bit環境双方でtest OK, 各desktop環境でtest OK
  - でも…

	GNOME	KDE	Xfce	LXDE
64bit	○	○	○	○
32bit	○	×	-	-

テスト未実施

でもトリアージが裏目に出ることが有ります。
一例ですがこんなことがありました。
64bit環境と32bit環境のtestに成功し、64bit環境上のGNOME, KDE, Xfce, LXDEでのtestに成功したので、もうDesktopまわりのbugは出ないものと思っていました。
しかし、testを省略した32bitのKDE環境にbugが潜んでいました。32bit環境でのみKDEのshellがcrashし、loginしても使えないという影響の大きいbugでした。

見逃さないためには

- みんなでやろう
- 手分けしてやろう
  - 見落とし・見逃しが潰せる
- 皆さん、一緒にテストしませんか?

トリアージによる見落としを避けるためにはみんなで手分けしてtestをすることが有効だと思います。
また、余裕があれば複数人で同じtestをやることでtestの精度を上げることができます。
皆さん、一緒にtestしましょう。

課題

- Server系のtest
  - 切り分けが難しい
    - Bug?
    - 設定miss? ←大概はこっち
- ガチガチに環境を規定して、その範囲内でtestする必要性?
  - ユースケースに合わせたtest
  - →企業内でのtest

つれづれtestについて説明してきましたが、このtestには弱点もあります。
Server・daemon系のtestはつれづれでは難しいのです。
いざ、おかしなことが起こったとき、それがbugなのか、設定ミスなのか、つれづれでは区別をつけるのが難しいです。Server・daemon系の場合、多くは単に設定を誤っていただけということが多いです。

Serverをtestするにはガチガチに環境を想定して、その範囲内でtestする、つまりtest設計をしっかりする必要があります。
業務でserverをいじっている方々は当然やってますよね?w

Bugreports

次にbugreportについて話をしたいと思います。

Bugが出たらreportすればいいのよ

- どこに
- 何を
- どうやって

Bugが出たらreportすればいいのですが、それが割と面倒くさいです。それをこれから解説していきたいと思います。

どこに

- 日本語forum (ex: Ubuntu日本語フォーラム)
- 日本語ML (ex: openSUSE-ja ML)
- distributionのbug-tracking-system (Bugzilla)等
- 英語ML (ex: openSUSE Factory ML)
- 上流(software開発元)のbug-tracking-system (ex: LibreOfficeのBugzilla)
- 上流のML (ex: gcrypt-devel ML)
- 等々

Bugの報告先はいろいろ考えられます。
日本語で報告したければforumやmailing listに報告することができます。
英語になりますがbug-tracking-systemがあれば、それを使えばいいでしょう。
mailing listも使えるかもしれません。
さらに上流のbug-tracking-systemやmailing listも報告先として考えられます。

どの順番で

- 日本語で議論できるところがあるならまずそこで。(forum, ML)
- Bug報告の必要性があるならbug-tracking-systemや英語MLへ
  - bugに違いないことが明らかな場合
  - 日本語forumで何の反応もない場合(;_;)
- 上流Softwareに報告する必要があるなら後から上流bug-tracknig-systemやMLへ
- この辺は事情により前後します
  - Bug report systemが自動起動する場合。
  - KDEなどはちょっと特殊

ではどの順番で報告すべきでしょうか?

日本語で議論できるところがあれば、まずそこで議論すればよいと思います。何しろ日本語で話ができて便利です。
その上でBug報告の必要があるならbug-tracking-systemへ上げればよいでしょう。英語に自信があるなら英語mailing listで議論してもいいと思います。
上流softwareのcommunityに報告する必要があるならdistributionへ報告してから上流のbug-tracking-systemやmailing listに報告しましょう。

Bug-report-systemが先に起動する場合はそちらへの報告が先になるかもしれません。(特にKDEはdistributionのbug-reportを飛ばしてreportする仕組みが有るのでそちらへの報告が先になるかもしれません。)

Accountを取ろう

- Bug-tracking-system siteとaccount取得pageの一覧

openSUSE	https://bugzilla.novell.com/index.cgi https://secure-www.novell.com/selfreg/jsp/createAccount.jsp
Fedora	https://bugzilla.redhat.com/ https://bugzilla.redhat.com/createaccount.cgi
CentOS	http://bugs.centos.org/main_page.php http://bugs.centos.org/signup_page.php
Ubuntu	https://bugs.launchpad.net/ubuntu/ https://login.launchpad.net/+new_account
Debian	http://www.debian.org/Bugs/ mailベースなのでaccount取得pageは無し

Bug-tracking-systemを使う上で面倒なのはaccountを取る必要があることです。ここでは主なdistributionのBug-tracking-system siteとaccount取得pageのURLを掲示しています。
赤いURLはBug-tracking-systemとAccount取得pageのsiteが異なるので注意が必要なところです。
openSUSEはNovellのSingle Sign Onが必要になります。

OS以外のbug-tracker-account

KDE	https://bugs.kde.org/ https://bugs.kde.org/createaccount.cgi
GNOME	https://bugzilla.gnome.org/ https://bugzilla.gnome.org/createaccount.cgi
LibreOffice	https://bugs.freedesktop.org/describecomponents.cgi?product=LibreOffice https://bugs.freedesktop.org/createaccount.cgi
VirtualBox	https://www.virtualbox.org/wiki/Bugtracker https://myprofile.oracle.com/EndUser/faces/profile/createUser.jspx

OS distribution以外の上流softwareのBug-tracking-system siteとaccount取得pageのURLも掲示しておきます。
私は一応これだけのaccountを取っています。
VirtualBoxは上流softwareとは言えませんが、重要なtest環境ですので、accountを登録しています。
VirtualBoxもOracleのSingle Sign Onが必要になります。

MLを購読しよう

- 本来はbugreportが目的じゃないMLも
  - でも他に相談するところがない
  - 不具合に関する相談ならありじゃない?
- あまりに沢山あるので省略(_ _)

Bug-tracking-systemのaccountだけでなく、mailing listも購読しておきましょう。
mailing listで注意すべきなのは本来がbugreportが目的ではないmailing listがあることです。
不具合に関する相談に留めておくのが無難かもしれません。

主要なmailing listを載せようかとも思ったのですが、数が膨大なのと、今述べたようにmailing listの目的がbugreportとは異なる場合が多いので割愛します。

既に報告されていないかチェック

- 既に報告されていても、あなたにはやることがある!
  - 自分の環境でも再現したという報告
    - もちろんあなたの環境を詳細に報告
  - 追加情報の提供
    - 再現条件
    - 再現率
    - logその他のdata

Bugらしき現象を見つけて、最初にすることは、そのbugが既に報告されていないかを調べることです。
Bug-tracking-systemで検索します。
既に報告済みの問題でも、自分が書き加えることで問題が進展するかもしれません。
自分の環境で再現したという事実と再現環境を書き加えましょう。
再現条件や再現率を書き加え、log等のdataがあれば添付します。

何を

- まずはbugを詳細に分析する
- 再現性を確かめる
- 繰り返し操作したらどうなるか?
- 同じ条件で再現するか?
- 条件を変えたら?(ex: VMware→VirtualBox)
- OSを変えてみる(openSUSE→Ubuntu)
  - 特定distributionのみの問題か、一般的な問題か?
- 版を変えてみる(openSUSE 13.1 →12.3)
  - デグレートか否か?

何を報告するかを決めるためにbugを詳細に分析します。
再現性があるか、繰り返し操作したらどうなるかを確認します。同じ条件や、違う条件で色々testしてみます。
余裕があるならOSを変えてみたり、版を変えてみたりしてtestします。
条件をいろいろ変えることで、再現条件が絞り込めます。

但し、今挙げたことを全部やっているととても時間がかかるのでcase by caseで対応します。

log,backtrace, etc...

- coreを吐いているならbacktraceを取ろう
- logを吐いているなら採取しよう。
- でもどんなアプリがどんなlogをどこに吐くんだろう?
- そこでこれらのpage
  - https://ja.opensuse.org/%E3%83%90%E3%82%B0%E3%83%AC%E3%83%9D%E3%83%BC%E3%83%88
  - https://en.opensuse.org/openSUSE:Submitting_bug_reports
  - https://wiki.ubuntu.com/DebuggingProcedures

Programがcoreを吐いていたならbacktraceを取ります。
coreの提出を求められることもあるので、coreは消さないで残しておいた方が良いかもしれません。
logを吐いているなら採取します。
問題は、applicationによって吐き出すlogが異なることです。
どんなlogをどうやって取ればよいかについては、詳しく解説されたpageがUbuntuやopenSUSEのsiteにあるのでご紹介しておきます。

例えばLibreOfficeならstraceを取るのが有効だとか、Firefoxでは逆にそれが役に立たない、などの情報が載っています。

backtraceの取り方(××の一つ覚えw)

- core を吐いたcommandの特定
  - file core
- Debug symbolの取得
  - Fedora, CentOS等の場合
    - debuginfo-install command
  - openSUSEの場合
    - gdb command core 2>err.txt
    - egrep '^Try: ' error.txt | sed 's/^Try: //' | sudo sh
  - Ubuntu・Debianの場合
    - 次で説明します。　→めんどくさいので、頑張れ!!!
- backtraceの取り方
  - LANG=C gdb command core 2>&1 | tee backtrace.txt
  - thread apply all backtrace full
- coreは残しておこう
  - 提出を依頼されることもあるので

backtraceを取るにはGDBを使う必要があります。でも難しくはありません。
まずfile coreでcoreを吐いたcommandを特定します。
command を特定したら、gdbにかけます。標準error出力にhintが現れるので、そのhintを元にdebuginfoをinstallします。
FedoraやCentOSなどYUM系のdistributionの場合はdebuginfo-install command一発でdebuginfoをinstall出来ます。
openSUSEの場合はgdbのerror出力を少し加工してやってshellに食わせる必要があります。
Ubuntu, Debianの場合は少しややこしいので後で説明します。
backtraceの取り方はgdbを起動し、全てのthreadに対してbacktraceを取るだけです。これは丸暗記しておいて損はないです。
ちなみにbug reportを出したcoreは残しておいた方が良いです。稀にですが提出を依頼される場合があります。

backtraceの取り方(Debian系の場合)

- 1. パッケージそのもののdbgシンボルを取る。
  - sudo apt-get install nautilus-dbg
- 1. GDB でbacktraceを取る。
  - $gdb /usr/bin/nautilus core 2>&1 | tee bt.txt
  - (gdb) thread apply all bt
- 2. backtrace からlibrary名一覧を抽出する。
  - $awk '/from/ { print $NF }' bt.txt | sort -u >libs.txt
- 3. library名以外のゴミを取り除く。
  - $vi libs.txt
- 4. library名からpackage名を引く。
  - $dpkg -S `cat libs.txt` | awk 'BEGIN { FS = ":" } { ++p[$1] } END { for (i in p) { print i }}' >pkg.txt
- 5. dbg packageをinstallする。
  - $sudo apt-get install `sed 's/$/-dbg/' pkg.txt`
- 6. 失敗するpackageがあるなら削除して、5.をやり直す。
  - $vi pkg.txt
  - $sudo apt-get install `sed 's/$/-dbg/' pkg.txt`
- https://forums.ubuntulinux.jp/viewtopic.php?id=15386

DebianやUbuntuの場合のdebug symbolの取り方です。commandが依存するlibraryの一覧をGDBで取ってきて、libraryがどのpackageに属するか調べて、そのpackageのdebug symbolを取ってきます。別に難しくはありませんが、正直面倒くさいです。

報告の形式

- titleは少々長くていいから具体的かつ詳細に…。
  - On Virtualbox, when I search "terminal" or "console", sometimes the gnome-shell crashes with SIGABRT in __GI_raise() at ../nptl/sysdeps/unix/sysv/linux/raise.c:56 .
- (どういう環境で)何々をしたら、こうなった。
  - (On 〜,) when〜, 〜
  - If 〜, then 〜
- 動作環境・packageのversion
  - 特にOSのversionは正確かつ詳細に
- 再現率
  - 必ず?5回に1回?、1日に5〜6回?…
- 具体的な再現手順
  - 箇条書きが楽
  - 命令形で
- こうなった
  - 次の「こうあるべき」と対比させるために書く
- でも本当はこうあるべきだよね。
  - 〜should〜

Bugreportの文章は大体形式が決まっています。
titleは少々長くていいから具体的にかつ詳細に書きます。「どういう環境や条件で」「何をしたら」「こうなった」という形式で書くと書きやすく、また解り易くなると思います。英語ではWhenやifを使えば簡単に書けると思います。但しbug-tracking-systemによっては、titleの長さが制限されている場合があるので、その場合には、本質的でないところを削って長さを合わせます。
本文でも「どういう環境や条件で」「何をしたら」「こうなった」を書きます。titleで書いたからといって、本文でその内容を省略するというようなことはしないでください。本文でもtitleで書かれた事を繰り返しさらに詳細に説明します。titleの長さ制限で書けなかったことも、本文では書けるので書きます。
bugreportの目的は開発者に再現手順を示すことです。「こうなった」という結果よりも「どういう環境や条件で」「何をしたら」の方を優先して詳細に書きます。
英語ならwhenやifを使えば文章が作り易くなると思います。
動作環境やpackageのversionを書きます。OSのversionは正確に書く必要があります。
再現率も書きましょう。必ず発生するのか、n回にm回なのか、t時間にn回なのか、具体的に書きます。
具体的な再現手順を箇条書きで書きます。英語なら命令形で書くと楽です。
起きた結果を簡潔に書きます。
最後に本来ならこう動くべきだという動作を書きます。英語なら「should」を使った文章を考えれば良いでしょう。

どうやって(1)

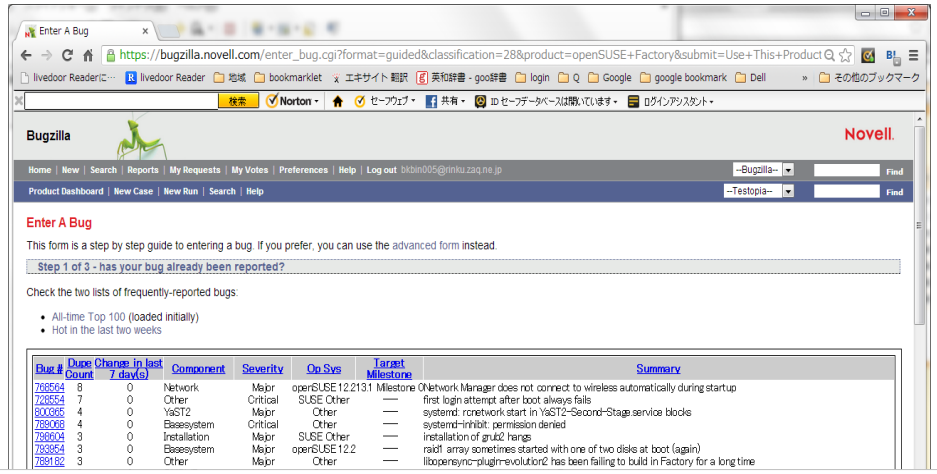
- ディストリビューションがbug reportの仕組みを持っている場合
  - それを使う

ディストリビューション	システム	コマンド
Ubuntu	Apport	ubuntu-bug
CentOS	ABRT	abrt-gui
Fedora(19)	ABRT	gnome-abrt
openSUSE	-	-
Debian	reportbug	reportbug

メジャーなdistributionはbug report systemを持っています。bug report systemを起動するcommandを掲載しておきます。bug report systemはapplicationの種類に応じて採取するlogを自動的に選択してくれるので便利です。
Bug report systemはprogramがcrashした場合、自動的に起動します。
便利な機能ですが、予想外のbugに出会って、急に入力画面が出てくると驚き、焦ります。GUIが壊れた場合などは入力するのが辛いこともありますが、落ち着いて入力する必要があります。
また、それぞれのcommandは使い方が微妙に異なるので、慣れておく必要があります。
メジャーなdistributionを調べてみましたが、openSUSEだけがbug report systemを持っていないようでした。残念です。

どうやって(2)

- ディストリビューションがbug reportの仕組みを持ってい無い場合
  - Bug tracking systemに直接login



openSUSE等のようにdistributionがbug report systemを持っていない場合は、bug-tracking-systemに直接loginしてbugを入力する必要があります。また、bug report systemを持っているdistributionでも、それを使えない場合というのがあります。例えばFedoraのABRTは、crashしたcommandしかreport出来ないと思います。
自分の使っているdistributionのbug report systemのaddressは覚えておくとよいでしょう。

英語の壁

- パターンで乗り切れ
  - 「報告の形式」参照
  - When〜, if〜,…
- 強い味方があるじゃないか
  - Excite
    - http://www.excite.co.jp/world/english/
  - Google 翻訳
  - Gmail
- 他の人のbug reportを読む、参考にする
  - nativeの人より、英語の下手そうな人のreportが参考になる
- より深い突込みが来たら
  - 何かしてくれ、と言われているようだが、なんだかよく解らないとき
    - What should I do concretely?
    - (具体的に何をすればいいんだい?)
  - 日本語MLで相談w

Bug reportで困るのは英語の壁があることです。言葉の壁でbug reportをためらってしまいがちです。でも英語ができなくてもbug reportは可能です。
英語を恐れることはありません。何とかなります。
まずpatternを活用することです。先ほど説明した「報告の形式」にあるpatternを活用してください。one patternでも一向に問題ありません。目的は相手にbugの存在と再現手順を伝えることです。それのみに注力してください。再現条件はwhenやifが使えると思います。
難しい文章を作成する必要が出たときや、解らない英語に出会ったときには翻訳サイトが役に立ちます。
私はExciteを使用しています。またgmailの翻訳機能もmailing listで役に立ちます。但しgoogleの翻訳はたまに否定文を肯定文として扱うなど、あべこべに訳すので注意が必要です。
他の人のbug reportを読んで、参考にしてください。このときnativeのこなれた英語よりも、自分のlevelにあった、英語の下手そうな人の英語の方が参考になります。mailアドレスやreporterの名前から国籍を推定できます。但し英語圏以外の人でも学歴が高い人はnativeに近い文章を書く場合があるので用心しましょう。
reportの後、返事が返ってきて、その返事の意味を取りかねることがあります。どうしても意味が解らない時には、正直に英語が下手であることを打ち明けて、具体的に自分が何をすべきであるのか指示を仰ぎましょう。
日本語mailing listで「〇〇のbug reportで返事が返ってきたんだけど何て言っているのかな?」と相談を持ちかけるのも手だとおもいます。

今後の課題

- 自分でpackageを修正できるようになれば…(野心)
  - その点openSUSEは敷居が低い(OBS: Open Build Service)

testとbugreportについてお話してきました
が、bugreportを出すことで終わりになるとは私は思っておりません。
本当はpackageの修正にも手を染めたいと思っています。
packageのメンテナンスについてはopenSUSEの
Open Build Serviceが敷居が低くて取りつきやすいと感じています。

まとめ

- とにかく行動
- 英語を恐れるな!
- testしよう。bugreport書こう
- softwareの品質が上がる
- あなたには勉強になる
  - しかも楽しい。
- 一石二鳥じゃね?