# I, Testbot

## Testing, with a little help from your friends

RALF HOLLY

There was a time when humanity faced the universe alone and without a friend. Now he has creatures to help him; stronger creatures than himself more faithful, more useful, and absolutely devoted to him. Mankind is no longer alone.
—Isaac Asimov, *I, Robot*

Almost all software engineering experts agree that continuous integration is superior to the "big-bang" integration approach employed in the dark ages. Especially agile methodologies such as Extreme Programming (XP) promote development processes that let developers add functionality little-by-little, thereby significantly reducing project risk.

To ensure that frequent code check-ins (so-called "code deliveries," or "deliveries," for short) are of high quality, frequent build and test cycles are inevitable. One such approach is the now famous "Daily Build and Smoke Test" process, described in detail in Steve McConnell's book *Rapid Development* (Microsoft Press, 1996). While daily building and smoke testing significantly reduces project risk, for large and complex projects you may want to take this approach to extremes by validating your code base on a check-in basis—that is, many times a day. The advantages are obvious. If you check in buggy code, it will be immediately detected, which means that there will be no more time-consuming "Who's broken the build?" quests.

In this article, I present an approach that puts the Herculean building and testing effort on the shoulders of a test robot, or "testbot."

*Ralf is principal of PERA Software Solutions and can be contacted at rholly@pera-software.com.*

### The Challenge

One of my current projects is a complex embedded-systems project. It isn't particularly large (around 20 developers and 100 KLOC), but it is a multiplatform, multifeature kind of project. At any time—or more precisely, within a rather short period of time—the system has to be in a releasable state for any hardware platform and various combinations of features supported by our common code base. Here, I refer to such a combination of features on a particular hardware platform as a "target."

Working on this project used to be like walking on a mine field. If you added a feature or fixed a bug for some target, there was a certain probability that you broke the build on another target or at least introduced a compiler warning. Over time, these issues tended to accumulate, resulting in a downward spiral. Lots of effort had to be expended later on to get back to a clean build.

Obviously, requiring the developers to verify all of the other platforms and products before delivering their changes would have taken way too much time, let alone the fact that not every developer had access to all of the toolchains (compilers, linkers, in-circuit emulators, and the like) that the code base supports.

### The Innards of a Testbot

I solved this problem by assigning dedicated testbots to all the targets we support. The testbots are workstations that host a program that listens on the source-code-versioning system (ClearCase, in our case) for new deliveries of developers. When a developer finishes delivering her task to the integration branch, the testbot executes the following steps:

1. Inform developer(s). To inform you that your check-in is under test, an e-mail is sent to you, plus a carbon-copy to the testbot administrator; see Figure 1. The e-mail details what is being tested (whose deliveries) and contains a link to the testbot's logfile. By inspecting the logfile, you can track the status of the testbot's work. In order to implement such an e-mail notification scheme, you need a username to e-mail address mapping. In our case, we use a simple two-column text file that even lets developers specify their mobile phone numbers. This enables the testbot to send you a short message via an e-mail-to-SMS gateway in case something went wrong (or everything built and tested fine).

"Testbots are workstations that host a program that listens on the source-code-versioning system"

2. Checkout code. The testbot checks out a private copy of the source code based on the last completed delivery.
3. Build the code. The whole code is built and possibly—as it is the case on our project—checked against compiler and Lint warnings.

4. Execute (smoke) tests. Depending on the size and the needs of the project, the testbot executes the tests. In our case, we restricted the test set to tests that only cover the most important nominal cases. If test execution consumes too much time, you might not be able to check enough code deliveries per day and hence will detect defects too late; that is, when the developers already went home.
5. Inform developer(s). The last step is sending another e-mail (and/or short message, if you want) to you, informing you about the outcome of the test. If the testbot spots a problem, it additionally sends a copy to all team members, warning them about problems on the integration branch; see Figure 2.

The testbot software was quite easy to implement. In total, it comprises less than 1000 lines of Perl code; however, the size depends on the kind of source-code-versioning system you are using— different systems offer different solutions for parallel software development and querying of check-in information. Listing One is the testbot pseudocode implementation. For simplicity, I've left out logging and error handling.

## The TX Model

A small disadvantage of the testbot I've just described is that it heavily relies on e-mail notification. Oftentimes, developers complain about the flooding of e-mails that the testbots send, just because some developer caused a Lint warning on a target they are only marginally interested in.

Thus, I crafted an extended version (see Listing Two) that differentiates between major and minor issues:

• Major issues. Major issues are problems that prevent other developers from integrating their code. We decided that compile- and link-time problems, as well as smoke test execution errors, belong to this category. Because of their severity, major issues are broadcast— just like before— by e-mail to all developers.
• Minor issues. Unlike major issues, minor issues do not prevent further code deliveries and hence are only reported by e-mail to the developer who delivered the code, the architect, and the project lead. On our project, compiler and Lint warnings belong to this category.

In addition to the e-mail-notification mechanism, there is a client program called "Observer" (implemented in Perl/Tk) running on all project member's workstations. The Observer monitors the state of all targets. The following information can be derived from Figure 3:

1. The last update to the screen occurred at 14:02:50 and happened due to a state change (start of build 504) reported by the testbot of the Venus target (indicated by the asterisk next to the name).
2. Because of the delivery of JOHNC et al. (you can get the full list of developers who delivered to the integration branch by placing your mouse cursor over "JOHNC…"), the Mars project currently has major issues. The previous run based on the check-in of PETERZ, however, was fine. Since the Mars testbot is idling, we can assume that JOHNC et al. haven't delivered a fix yet; hopefully, they are working on it.
3. JOHNC et al. didn't cause any trouble to the Jupiter target.
4. Saturn is still being tested, but the previous run (build 473) reported minor issues. Most likely, JACKF just fixed these issues because he is also the originator of the current testbot run (build 474).
5. Pluto and Venus are still being tested. The previous run was fine.

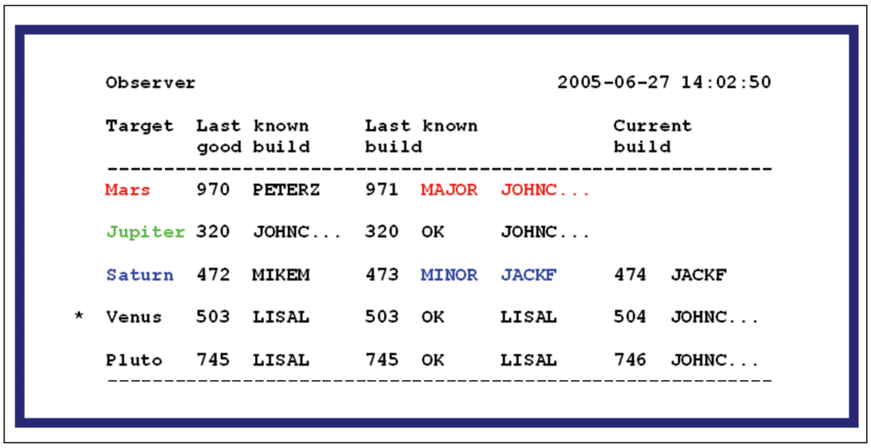
To provide Observer with build and test state information, the testbots drop cookies on a shared file server directory. Figure 4 presents the contents of the cookie files for a particular target. The Observer regularly checks this shared directory for

```
From: Eddie the Testbot
To:   JOHNC, GARYW
CC:   RALFH
Subj: [Mars] Checking build 971

Hi there!

I've just started checking build 971.

Between my last successful build (970) and this build the
following branches have been delivered:

ab-cleanup-lint-warnings by JOHNC on 2005-06-27 12:58:00
ab-optimize-network-interface by GARYW on 2005-06-27 13:37:50

The progress/results of this build can be found in
file:\\eddie\eddiepublic\eddielogs\eddielog971.txt


Cheers,
Eddie (your friendly Testbot for the Mars target)

PS: If you don't want to receive my emails in the future,
remove your name from
file:\\eddie\eddiepublic\user2emaillist.txt
```

***Figure 1:*** *E-mail notifying developers JOHNC and GARYW that their check-ins are under test.*

```
From: Eddie the Testbot
To:   ALL
CC:   RALFH
Subj: [Mars] Link-time error

Bad news, folks!

I'm having trouble with build 971.

Between my last successful build (970) and this build the
following branches have been delivered:

ab-cleanup-lint-warnings by JOHNC on 2005-06-27 12:58:00
ab-optimize-network-interface by GARYW on 2005-06-27 13:37:50

The progress/results of this build can be found in
file:\\eddie\eddiepublic\eddielogs\eddielog971.txt

Can the branch owners please have a look at this logfile?

Cheers,
Eddie (your friendly Testbot for the Mars target)

PS: If you don't want to receive my emails in the future,
remove your name from
file:\\eddie\eddiepublic\user2emaillist.txt
```

***Figure 2:*** *E-mail informing the whole team about problems related to JOHNC and GARYW's check-ins.*



```
     Observer                                       2005-06-27 14:02:50

     Target   Last known      Last known            Current
              good build      build                 build
     -----------------------------------------------------------
     Mars     970  PETERZ     971  MAJOR  JOHNC...

     Jupiter  320  JOHNC...   320  OK     JOHNC...

     Saturn   472  MIKEM      473  MINOR  JACKF     474  JACKF

   * Venus    503  LISAL      503  OK     LISAL     504  JOHNC...

     Pluto    745  LISAL      745  OK     LISAL     746  JOHNC...
     -----------------------------------------------------------
```

***Figure 3:*** *Data displayed by the Observer tool.*

```
 ; contents of mars.current (current build info)
state=null
text=null
build=null
users=null
time=null

; contents of mars.lkb (last-known build info)
state=major
text="Link-time error"
build=971
users=JOHNC,GARYW
time=2005-06-27 13:47

; contents of mars.lkgb (last-known good build info)
state=OK
text="OK"
build=972
users=PETERZ
time=2005-06-26 18:47
```

***Figure 4:*** *The contents of the Mars testbot's cookie files.*

changes and displays the information in a user-friendly dialog.

By employing this mixed push/pull (e-mail/Observer) scheme, we significantly reduced the number of e-mail broadcasts, thus improving developer happiness. Moreover, the Observer is a nice tool for project leads because they get constant feedback on the quality of their and other leads' projects.

## The Sky Is the Limit

Once you have build and test automation, as well as a testbot framework in place, you can assign all kinds of useful tasks to your testbots. Why not have a testbot execute all of your system tests, stress tests, and performance measurements many more times than you used to? On our project, we have a testbot that takes care of executing all of the thousands of developer (unit) tests. It iterates over a list to which developers add references to their developer test suites. Of course, these extended tests take days to execute, implying that you cannot test individual check-ins but rather whole sets of check-ins. Still, if something goes wrong, you know exactly who made changes to the code base and who is likely to be the culprit.

## Conclusion

Delivering functionality in small increments is the preferred way of developing systems these days. Especially on medium to large projects, testbots help ensure code quality and reduce project risk through frequent testing—much more testing than any human being is able to bear. Equip your testbot with a friendly UI and it will be, as the Sirius Cybernetics Corporation would put it, "your plastic pal who's fun to be with."

**DDJ**

## CruiseControl

Shortly after I had finished this article, the open-source project CruiseControl (http://cruisecontrol.sourceforge.net/) was brought to my attention. So far, I haven't had a chance to use it in a project, but from what I've seen on their web site, it looks like a solid tool to me. It comes with e-mail notification support and reporting facilities. Before you reinvent the (build)loop, you might want to give CruiseControl a try. Let me know what you think of it.

—R.H.

## Listing One

```
loop forever
    sleep for 5 minutes
    check versioning system for new deliveries
    if new deliveries since last testbot run
        get list of developers who delivered
        foreach developer who delivered
            send 'test started' email
        checkout code based on last delivery
        execute build
        if build problems
            foreach team member
                send 'build problems' email
        redo loop
        execute smoke test
        if smoke test problems
            foreach team member
                send 'smoke test problems' email
        redo loop
        foreach developer who delivered
            send 'test successful' email
redo loop
```

## Listing Two

```
loop forever
    sleep for 5 minutes
    check versioning system for new deliveries
    if new deliveries since last testbot run
        get list of developers who delivered
        drop 'build in progress' cookie (target.current)
        foreach developer who delivered
            send 'test started' email
        checkout code based on last delivery
        execute build
        if build major problems
            drop 'build major problems' cookie (target.lkb)
            foreach team member
                send 'build problems' email
        redo loop
        if build minor problems
            drop 'build minor problems' cookie (target.lkb)
            foreach team member, architect, project lead
                send 'build minor problems' email
        redo loop
        execute smoke test
        if smoke test problems
            drop 'smoke test problems' cookie (target.lkb)
            foreach team member
                send 'smoke test problems' email
        redo loop
        drop 'test successful' cookie (target.lkgb)
        remove 'build in progress' cookie (target.current)
        foreach developer who delivered
            send 'test successful' email
redo loop
```

**DDJ**