Project Automation

If it hurts, automate it!

Jan Pool

NioCAD – University of Stellenbosch

19 March 2008

Introduction

- Purpose:
  - Introduce various aspects of project automation.
  - Why, when, what, and how to automate?
- Target Audience:
  - Developers.
  - Project Managers.
- Disclaimer:
  - I’m still a student of the art of automation.
  - Automation will not solve all your problems.
    - Remember “No Silver Bullets” – Frederick P. Brooks.

Sources

- Sources:
  - Pragmatic Project Automation:
    - Mike Clark
    - ISBN-10: 0974514039
    - http://www.pragprog.com/titles/auto
    - Cost about $20 at Amazon.
  - Pragmatic Programmer:
    - Andrew Hunt and David Thomas
    - ISBN-10: 020161622X
    - http://pragmaticprogrammer.com/the-pragmatic-programmer
    - Cost about $38 at Amazon.
  - Practices of an Agile Developer – Working in the Real World
    - Venkat Subramaniam and Andy Hunt
    - http://www.pragmaticprogrammer.com/titles/pad/
    - ISBN: 0-9745140-8-X
    - Cost about $20 at Amazon
  - Sweat equity.

Why Automate?

- Improve consistency and repeatability.
- Automated processes is less error prone.
  - People hate repetitive and boring tasks.
- Gain and maintain confidence in product quality level.
- Repeat critical procedures more often.
- Become aware of, and fix problems earlier.
  - Easier to fix problems, since little problems do not compound.
  - Easier to isolate offending code.
- Improve feedback and continuously know the state of the project.
- Reduces documentation.
- "Make your developments more precise with automation.“
  - Quote from Pragmatic Programmer book page on Amazon.

When to Automate?

- Start automation as early as possible.
- Automate when:
  - You feel the pain,
  - it will save time,
  - performing a task manually is error prone,
  - a task is repetitive,
  - a task is brain-dead-boring,
  - a task is critical and should be performed regularly.
- Remember, automation takes time to setup and to maintain.
  - Do not automate for the sake of automation.
  - Justify time investment with possible gains.
  - Use existing technologies and investments where you can.
- Rule of thumb: if you run it more than twice, automate it.

What to Automate?

- Testing: Unit, Integration, Acceptance, etc.
  - Without automated tests, you have no confidence in the quality of your automated builds.
  - Code coverage helps to guide testing effort.
  - Code analysers helps to detect common problems.
- Build process.
- Deployment.
- Backups.
- Information that changes predictably (use code generators).
- Other possibilities?
  - Workflows.
  - Perhaps schedule code reviews.

How to Automate?

- Covered in the rest of the talk.
- Types of automation:
  - Commanded automation.
    - Run a command.
  - Scheduled automation.
    - Run commands from a scheduler so that it executes automatically.
  - Triggered automation.
    - Run command based on triggers (ex. version control check-ins).
- Requires:
  - Version control.
  - Scripting.
  - Communication: email, sms, rss, web pages, visual aids, etc.

Testing

- Unit and Integration testing:
  - xUnit frameworks (JUnit, NUnit, pyUnit, CPPUnit, etc.)
- Acceptance testing:
  - FitNesse (http://fitnesse.org/)
- GUI testing:
  - Squish, TestComplete, Eggplant, etc.

One-step Build and Test

- Build and test your project in one step.
- Builds should be CRISP:
  - Complete:
    - Build from scratch and independently without human intervention.
  - Repeatable:
    - Must be able to create exactly the same build at a later time.
    - Store build scripts in source control.
  - Informative:
    - "Detector of unexpected changes".
    - Provide information on why a build failed.
  - Scheduled:
    - Let the builds run automatically.
  - Portable:
    - Build should be runnable from any system (same platform), not just that of the developer.
    - For cross-platform software, it should build on all platforms.

One-step Build and Test (cont.)

- IDEs and CRISP:
  - Can be difficult to get CRISP builds through an IDE.
  - More IDEs provide an automation interface.
- Build directory:
  - Keep source and test code separate for easier management.
  - Keep third-party libraries in source control to ensure the correct version is available.
    - Ensures complete and repeatable.
  - Separate source, configuration, docs and build outputs.
    - Outputs are not stored in VC.
- Script your builds:
  - Ant, NAnt, Maven, MSBuild, Make, SCons, custom, etc.
  - Makes it repeatable.
- Testing:
  - Perform testing as part of build.
  - Generate reports.
  - Fix test when they break!

Schedule Builds

- Take the One-Step build and schedule it.
- Continuous builds using continuous integration software.
- Use frequent incremental builds to provide quick feedback.
  - Use check-in triggers.
  - But do a clean build every now and then (ex. nightly).
- Write custom script or use applications such as CruiseControl.
  - Use build scripts to do all the hard lifting.
  - Enables one to test the whole build from the command line using ant.
- Integrate test, test coverage and code analysis results with scheduled build reports.
- Publish results, artifacts and build status:
  - Email, RSS, IM, SMS, visual aids, etc.
- How often?
  - Depends on task.
  - Builds should be frequent enough to give feedback on system health in good time (check-ins).
  - Releases and deployment less often: nightly, iteration, monthly, etc.

Push-button Releases

- Create a release on command.
- Create branches to shield the release from further commits.
- Checkout and test release branch.
- Package release (installer).
- Package optional test files separately.

Installation & Deployment

- Install or deploy to a test system.
  - Use virtualisation.
- Test installs in isolation on clean machines.
- Troubleshoot using diagnostic tests.

Monitor Automation Scripts

- Use triggered automation to support monitoring.
- Running applications.
  - Monitor log files of running apps to catch problems.
- Feedback via Email, RSS, IM, etc.
  - But what if you are not at your workstation?
- Use SMS for critical errors.
- Visual devices, but must be:
  - Unique,
  - attracts attention,
  - hard to miss and
  - very visible (near team).

Cross Reference

- Practices of an Agile Developer:
  - Practice #13: Keep it releasable.
  - Practice #14: Integrate early, integrate often.
  - Practice #15: Automate deployment early.
  - Practice #19: Put angels on your shoulders.
  - Practice #22: Automate acceptance testing.
  - Practice #27: Actively evaluate trade-offs.

Cross Reference

- Pragmatic Programmer:
  - DRY—Don't Repeat Yourself.
    - Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
  - Write Code That Writes Code.
    - Code generators increase your productivity and help avoid duplication.
  - Don't Use Manual Procedures
    - A shell script or batch file will execute the same instructions, in the same order, time after time.
  - Coding Ain't Done 'Til All the Tests Run.
  - Test State Coverage, Not Code Coverage.
    - Identify and test significant program states. Just testing lines of code isn't enough.
  - Test Early. Test Often. Test Automatically.
    - Tests that run with every build are much more effective than test plans that sit on a shelf.
  - Find Bugs Once.
    - Once a human tester finds a bug, it should be the last time a human tester finds that bug. Automatic tests should check for it from then on.

Final Comments

- Start small with your automation effort and improve incrementally.
- Only spend time on automation if you feel the pain or can see the reward.
- Everyone on the team should be involved with automation tasks.

Questions?

- Questions?
- Many more Pragmatic titles at:
  - http://www.pragmaticprogrammer.com/
- Contact Information:
  - jpool-at-pshymorphic.com
  - http://www.pshymorphic.com/
  - NioCAD: http://research.ee.sun.ac.za/niocad/