$Id: pair-programming.mm,v 1.21 2014-01-03 17:11:48-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Syllabus/pair-programming
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Syllabus/pair-programming/

1. Pair Programming

Pair programming is a software development technique in which two programmers work together at one keyboard. One types in code while the other reviews each line of code as it's typed in. The person typing is called the ***driver***. The person reviewing the code is called the ***observer*** or ***navigator***. The two programmers switch roles frequently (possibly every 30 minutes).

While reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely future problems to address. This frees the driver to focus all of his or her attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

Some benefits of pair programming are : design quality, learning and training, overvoming difficult problems. You have another head to bounce problems off of without cheating. Some drawbacks are : work preferences, difficulties in scheduling meetings, working with an uncooperative partner.

2. Guidelines

If you wish to do so, you may use pair programming to develop your programs. Following are some guidelines :

(a) Pair programming is not compulsory. You may choose a pair partner, or you may work independently.

(b) Before choosing a partner, interview each other to verify that he or she has the necessary knowledge and that your levels of accomplishment are roughlly the same. An A+ student will generally prefer to work with another A+ student, but will not get a lot of help from a C– student. You are entirely responsible for you choice of partner, or for you choice of working alone.

(c) Spend 75% of your development time working together at the same workstation.

(d) After working alone each partner should review each line added by the other partner. Make use of **`diff`**(1) or **`rcsdiff`**(1) to check up on changes between various versions.

(e) Alternate between driving and observing approximately after every 30 minutes of programming time.

(f) One partner submits the code and the other watches to verify that the correct code has been submitted. You might make use of the script **`testsubmit`**.

3. Requirements

In addition to the requirements for students working alone, pair programming teams have a few additional requirements.

(a) Both partners must submit a **`README`** and a **`PARTNER`** file, but only one partner should submit the code.

```
 1
 2  This program was completed using pair programming.
 3  Partner:  Ada Byron (abyron@ucsc.edu)
 4  Partner:  Charles Babbage (cbabbage@ucsc.edu)
 5
 6  We acknowledge that each partner in a programming pair should
 7  drive roughly 50% of the time the pair is working together, and
 8  at most 25% of an individual's effort for an assignment should
 9  be spent working alone.  Any work done by a solitary programmer
10  must be reviewed by the partner.  The object is to work
11  together, learning from each other, not to divide the work into
12  two pieces with each partner working on a different piece.
13
14  Ada Byron spent       ____ hours working alone.
15  Charles Babbage spent ____ hours working alone.
16  We spent              ____ hours working together.
17  Ada Byron spent       ____ hours driving.
18  Charles Babbage spent ____ hours driving.
19
20  Please grade the work submitted by abyron@ucsc.edu
21  and not the work submitted by cbabbage@ucsc.edu.
22
23  $Id: README,v 1.1 2011-08-22 13:05:55-07 - - $
```

Figure 1. Pair programming `README`

(b) Both partners name and username must appear in a comment at or near to top of ***every*** file submitted, except for the **`PARTNER`** file.

(c) If both partners have submitted the project, then both partners must state in their **`README`** which of the submissions is to be graded. Only one partner's work will be graded.

(d) The **`README`** submitted by each partner must be the same and must agree. See Figure 1 for a sample of what it should look like. Substitute your names for Ada and Charles. After the required notes shown in Figure 1 that the grader should read before grading your work. The line numbers are not actually present in the file ; they are generated by this text formatter.

(e) The **`PARTNER`** file must contain one single word, namely the partner's username. So Ada should submit a **`PARTNER`** file containing the word **`cbabbage`** and Charles should submit a **`PARTNER`** file containg the word **`abyron`**. This is most easily done from the command line rather than using an editor. Ada creates her **`PARTNER`** file with the command

```
echo cbabbage >PARTNER
```

Charles creates his **`PARTNER`** file with the command

```
echo abyron >PARTNER
```

Use the script **`bin/partnercheck`** to verify correctness. Obviously these files can not be kept in the common working directory. The **`PARTNER`** file is used to

```
 1
 2  Grader: If the student is working alone, ignore this file and
 3  do not deduct any points described here.  If the students are
 4  working in a pair, grade *ONLY* one of the pair's work.
 5
 6  Make a *relative* symlink in the non-graded partner's directory
 7  called SCORE pointing at the SCORE file in this directory.  Do
 8  not copy the SCORE file.  Create the symlink via:
 9  .   ln -s ../(partner)/SCORE
10  where (partner) is the partner's actual username.  After that,
11  ls -la should show something like
12  .   lrwxr-xr-x   17 Jun 10 18:04 SCORE -> ../(partner)/SCORE
13
14  (3) Did both partners' names and usernames appear in a comment
15      at or near the top of all text files submitted (except
16      see below for PARTNER file)?
17
18  (3) Did they both submit a README which contained the required
19      pair programming log?
20
21  (4) Was the PARTNER file exactly correctly formatted?
22      Use partnercheck to verify this.  Both partners must have
23      a PARTNER file pointing at each other for either of them
24      to get these points.  Both users' PARTNER files must be
25      reported as valid by the partnercheck script.
26
27  These files must be spelled README and PARTNER, respectively,
28  in exactly that way, in upper case.  Add up the negative points
29  above and enter that on the TOTAL1 line.
30
31  TOTAL1=
32  .between -10 and 0, inclusive.
33
34  If this is a project graded out of 60 points, reduce TOTALSCORE
35  in the SCORE file by TOTAL1.  Ignore TOTAL2.  If this is a lab
36  graded out of 30 points, TOTAL2 = TOTAL1 / 2, using integer
37  division, and reduce the TOTALSCORE by TOTAL2.
38
39  TOTAL2=
40  .between -5 and 0, inclusive.
41
42  $Id: SCORE.pair,v 1.5 2011-08-23 15:46:54-07 - - $
```

Figure 2. `SCORE.pair`

match partners and automatically enter the score into both partners' grade file

line items.

(f) ***Common error:*** Note that the `PARTNER` file must contain your CruzID and a new line character, and nothing else — not a personal name, not an entire email address. It is not read by the grader — it is read by a script that matches partners.

(g) The `partnercheck` script verifies the validity of the `PARTNER` file. Make sure that the directory `/afs/cats.ucsc.edu/courses/cmps012b-wm/bin` is in your `$PATH`.

(h) If you and your partner come up with irreconcilable differences and can no longer work together, explain this clearly in the `README`, each separately, and do not work together again. Each should then submit work separately.

(i) In the case of breakup, do not submit a `PARTNER` file. Since you can not delete a file once submitted, if you have already submitted on, resubmit a `PARTNER` file with zero bytes in it. This can be created via :

```
cp /dev/null PARTNER
```

(j) Pair programming code will be graded in the usual way by the grader, but additional points will be deducted for each of the following :

(i) Both partners' names and usernames were not in a comment at or near the top of each file submitted.

(ii) The `README` file did not contain the required information.

(iii) The `PARTNER` file was not exactly correctly formatted.

(iv) In the case of a team breakup, the reasons were not documented in the `README`.

(k) ***"But I thought that my partner submitted the code."*** That is absolutely not an acceptable excuse. When you are doing pair programming both partners are sitting together at the same workstation and it is the observer's duty to verify that the driver actually did submit the correct version of the code.

(l) Look at the file `SCORE.pair`, shown in Figure 2, which the grader will fill in along with the `SCORE` file. Make sure to verify that you have done what is required.

4. Cheating

Doing pair programming and not acknowledging it is ***academic misconduct***. See the syllabus for penalties. Pair programming is encouraged, so each partner should verify that both names and usernames are in all files submitted. If you are ever unsure, resubmit a file. Only the most recent version of any file will be examined by the grader.

5. Sharing files

When you are working together at the same computer, you will be logged into one or the other's account. If you are working separately on occasion, one partner might want to access files in the other partner's account. We will assume again that the partners are `abyron` and `cbabbage`.

(a) You should keep a common codebase in one or the other's account. Perhaps use `fs lq` to determine who has the most free disk space.

(b) Say they want to use Ada's account as a code base. Ada types :

(i) Create a shared directory.
```
mkdir $HOME/shared
```

(ii) Allow Charles to list the home directory.
```
fs sa $HOME cbabbage l
```

(iii) Make sure only Ada can access the shared directory. This deletes any previous ACLs that might be inappropriate.
```
fs sa $HOME/shared abyron all -clear
```

(iv) Also allow Babbage to access this directory.
```
fs sa $HOME/shared cbabbage write
```

(v) The newly created working directory inherits the ACL from `shared`.
```
mkdir $HOME/shared/cmps012a
```

(c) You should make use of a code archival system such as RCS, CVS, or SVN. Or you can just make backup copies using `cp`.

(d) Each partner should regularly make a recursive copy of the common code base, just in case one partner deletes everything, and so that each has copies of the code after the end of the quarter, when Byron will likely remove Babbage's ACL.

(i) Type the command `pwd` in the directory where you have set up your shared code.

(ii) Make a symbolic link from your private directory to the shared code. In your private directory, type
```
ln -s /afs/cats.ucsc.edu/users/——— codebase
```
Note that the middle operand has a line in it indicating that you should actually replace it with the exact output of `pwd` from the shared directory.

(iii) In order to quickly get from anywhere to the shared code base, put alias commands in your shell startup file. If you use `bash`, add the following line to your `.bashrc` :
```
alias cdp="cd /afs/cats.ucsc.edu/users/———"
```
If you use `csh` or `tcsh`, add the following line to your `.cshrc` :
```
alias cdp cd /afs/cats.ucsc.edu/users/———
```

6. References

You might want to look at some of the following references :

(a) The Wikipedia has a description of pair programming.
`http://en.wikipedia.org/wiki/Pair_programming`

(b) Charlie McDowell also has a collection of references to pair programming.
`http://www.soe.ucsc.edu/classes/cmps012a/Fall08//supplements/pairProgramming.html`

(c) UCSC ITS has a description of AFS and access control lists, which you need to understand.
`http://its.ucsc.edu/services/web/unix/afs/`