## Thursday, October 13

## Announcements

- Read No Silver Bullet by Brooks for Tuesday

## Levels of Ignorance

## Importance of Ignorance

- Everyone is ignorant about something-- often many things
- The first step in becoming an intelligent ignoramus is to understand ignorance

## Armour’s Orders of Ignorance

- Zeroth Order Ignorance (0OI): Lack of ignorance.
  - I have 0OI when I provably know something.
- First Order Ignorance (1OI): Lack of knowledge.
  - I have 1OI when I do not know something.
- Second Order Ignorance (2OI): Lack of awareness.
  - I have 2OI when I do not know that I do not know something.
- Third Order Ignorance (3OI): Lack of Process.
  - I have 3OI when I do not know of a suitably efficient way to find out that I do not know that I do not know something.
- Fourth Order Ignorance (4OI): Meta ignorance.
  - I have 4OI when I do not know about the Five Orders of Ignorance.

## Zeroth Order Ignorance (0OI)

- Lack of Ignorance
  - I have Zeroth Order Ignorance (0OI) when I know something and can demonstrate my lack of ignorance in some tangible form.
  - 0OI is provable and proven knowledge that is deemed “correct” by some qualified agency. In software this means that the knowledge is invariably factored into usable form. In all forms of knowledge there must be some external “proof” element that qualifies the knowledge as being correct.
  - Examples
    - Trivia
    - Building a system that satisfies the user
    - Ability to sail

## First Order Ignorance (1OI)

- Lack of Knowledge
  - I have First Order Ignorance (1OI) when I do not know something and I can readily identify that fact. 1OI is basic ignorance or lack of knowledge.
  - Example
    - Speaking Russian

## Second Order Ignorance (2OI)

- Lack of Awareness
  - I have Second Order Ignorance (2OI) when I do not know that I do not know something. That is to say, not only am I ignorant of something (I have 1OI), I am unaware of what it is I am ignorant about. I do not know enough to know what it is that I do not know.
  - Example
    - I cannot give a good example of 2OI, of course.
    - Do you need a will or a trust?

## Third Order Ignorance (3OI)

- Lack of Process
  - I have Third Order Ignorance (3OI) when I do not know of a suitably efficient way to find out that I do not know that I do not know something, which is lack of a suitable knowledge-gathering process.
  - This presents me with a major problem: If I have 3OI, I do not know of a way to find out that there are things that I do not know that I do not know. Therefore, I cannot change those things that I do not know that I do not know into either things that I know, or at least things that I know that I do not know, as a step toward converting the things that I know that I do not know into things that I know.
  - Example
    - Design
    - Investigative journalism

## Fourth Order Ignorance (4OI):

- Meta Ignorance
  - I have Fourth Order Ignorance (4OI) when I do not know about the Five Orders of Ignorance.
  - I do not have this kind of ignorance, and now neither do you.
  - Knowledge is highly and intrinsically recursive-- to know about anything, you must first know about other things which define what you know.

## Asking Questions

- Reveals
  - Ignorance
  - Intelligence
- Reduces
  - Ignorance
  - Assumptions
- Examples
  - Richard Feynman
  - Dan Berry

## Feynman: Oak Ridge Facility

## Berry: Ignorance is Key

- It was clear to Dan that the main problem preventing the engineers at the start-up from coming together to write a requirements document was that
  - All were using the same vocabulary in slightly different ways,
  - None was aware of any other's tacit assumptions,and
  - Each was wallowing deep in his own pit.

## Berry's Claim: Need Ignorance

- Our conclusion is that every requirements engineering team requires a person who is ignorant in the application domain, the Ignoramus of the team, who is not afraid to ask questions that show his or her ignorance, and who will ask questions about anything that is not entirely clear.

## References

- Philip G. Armour (2003) The Laws of Software Process: A New Model for the Production and Management of Software, Auerbach Publications.
- Richard P. Feynman and Ralph Leighton (1984) Surely You're Joking, Mr. Feynman! (Adventures of a Curious Character), W W Norton & Co Inc.
- Daniel M. Berry (1995) “The Importance of Ignorance in Requirements Engineering,” Journal of Systems and Software, 28:2, 179–184, February.

## Reducing 0OI

- 0OI: I have the answer.
  - Developing a system is simply a matter of transcribing what I already know, into the appropriate programming instructions. Note that this is not possible unless I know the appropriate coding medium-- that is, I know how to program. Of course, not knowing the language, not knowing how to program, is actually a form of ignorance, which I would presumably know about. If that were the case, I would classify it as a form of 1OI to be dealt with below.

## Reducing 1OI

- I have the question
  - I either know how to get the answer to it (I have 0OI about the activity of answer acquisition) or I do not know how to get the answer (I have 1OI about answer acquisition).
  - We can classify very high-level, open questions (such as, “What are the requirements for this system?”) as being type-questions. They are the typical questions posed by meta processes. They are generic, vague, context-free, and domain nonspecific. They may also not be very useful in some situations because they will often generate vague (context-free) answers. While such questions are helpful when I do not have a context to ask specific questions, they do not usually produce usable engineering answers. The most useful output from a type-question is often another more-specific question. If I find I am using many type-questions, it usually indicates that I have significant amounts of 2OI.

## Reducing 2OI

- I do not have the question. I do not know enough to frame a question that is contextual enough to elicit a definitive answer.
  - I may fall back on type-questions or some other mechanism. This fall-back mechanism is a “Third Order Ignorance Process.” I will argue that most, if not all, software methods are 3OI processes. Some types of systems are predominantly 0OI and 1OI heavy and others have large amounts of 2OI.
    - Research
    - Porting software
  - But in any systems development there is always some measure of 2OI, and we must deal with this in a different way than 1OI. This means we have to adopt a different process for each kind of unknown and different processes for 0OI/1OI and 2OI/3OI.

## Reducing 3OI

- Understand the limitations of process.
  - Detailed, granular processes are the most useful, although they are usually only useful across restricted problem spaces. The reason they are useful is that the processes contain the context of the problem. I call high-level, general processes that do not contain the context meta processes. They are usually not very useful in obtaining definitive answers and instead tend to generate more questions.
    - Waterfall: Requirements, Design, Code, Test
  - The process at the life cycle level does not tell us exactly what to do, and when and where to do it. It does not tell us what kind of requirements to get and from whom. This is because generically we do not know what to do that would be effective; because we do not know what questions we are trying to answer, until we start operating on this system. This limitation is inherent to all processes.

## Reducing 4OI

- Understand the knowledge acquisition activity.
  - The global application of the same methodology regardless of what is known or not known of the problem, or what kind of problem it is, could be viewed as 4OI in operation. The squandering of hard-won knowledge assets is another example of this level of ignorance in operation.

## Armour's Orders of Ignorance

- Zeroth Order Ignorance (0OI):  Lack of ignorance.
  - I have 0OI when I provably know something.
- First Order Ignorance (1OI):  Lack of knowledge.
  - I have 1OI when I do not know something.
- Second Order Ignorance (2OI):  Lack of awareness.
  - I have 2OI when I do not know that I do not know something.
- Third Order Ignorance (3OI):  Lack of Process.
  - I have 3OI when I do not know of a suitably efficient way to find out that I do not know that I do not know something.
- Fourth Order Ignorance (4OI):  Meta ignorance.
  - I have 4OI when I do not know about the Five Orders of Ignorance.

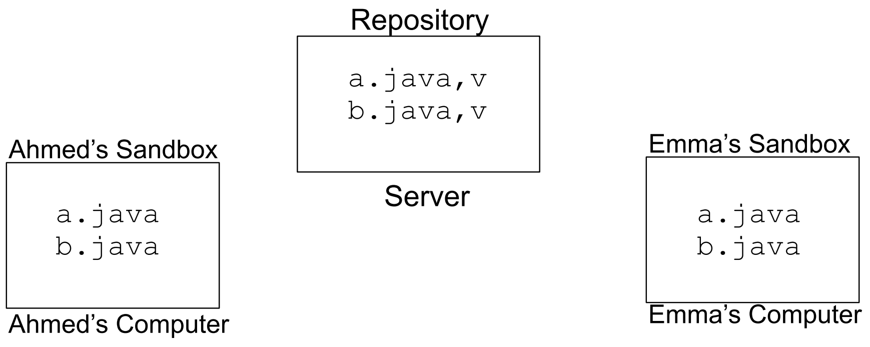
## Configuration Management (Version Control)

## Version Control Tools

- Provide a complete history of a file
  - All current and previous versions available- nothing ever goes away
- Other names
  - Revision Control System
  - Source Control
  - Configuration Management
- Idea:
  - Keep files in repository
  - Also keep a history of the changes to the file
  - Use the history to recover any version of the file
  - Also record times: "What did these files look like at noon last Friday?“
  - Can also tag groups of files

## How Version Control Works

- Place the official version of source code into a central repository, or database
- Programmers check out a working copy into their personal sandbox or working copy
- When finished and fully tested, programmers check in their code back to the repository



## Subversion

- subversion
  - Open Source
  - Available since 2000 as a successor to CVS
  - Suitable for individuals or medium-sized teams, though large teams are using it too
  - Runs on Linux, Solaris, Mac OS X, Windows, and others
  - http://subversion.tigris.org/
- On command line, check in and check out accomplished by typing commands
- GUI front ends/clients available
  - TortoiseSVN, Eclipse plug-in subclipse

## subversion Commands

- checkout
  - Retrieving a file from the repository
- commit
  - Putting changes back into repository
- update
  - Refresh the local or working copy copy with any changes since checkout
- http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf

## Control Regimes

- Pessimistic Model
  - File becomes locked on check out
  - Nobody else can make changes
- Optimistic Model
  - File not locked on check out
  - Conflicts or collisions are managed at check in

- Default in subversion is optimistic model, but pessimistic model possible

## Conflict Detection and Management

- On check in, official repository copy is compared with new copy
  - Check version number
    - If repository and working copy versions are the same, accept the changes
    - If repository version is newer, reject commit operation.
- Need to update working copy before check in
  - Includes a synchronization step to merge changes from two files (new repository version and modified file from working copy)

## Conflict Detection and Management

- Merge algorithm
  - Line by line comparison
  - Changes to different lines are OK
  - Changes to same lines labeled as a conflict
    - Both versions written to the working copy copy
    - Choose which line by editing manually
  - No guarantee of code correctness after merge
- On a successful check in, save new version of files with a set of backwards references to changes
- Can apply detection selectively
  - Binary files only use version numbers or timestamps
  - No conflict detection applied to files that are ignored

## Operations are Atomic

- Checkout, update, and commit operations are like database transactions, they are all or nothing
  - In case of network error, machine crash, etc.
  - Avoids partial operations
- Operations work on directories
  - Recursively applied to files
- Every commit transaction is assigned a number, indicating a version of the directory
  - In other words, Version N and Version M of a file may be the same

## subversion Commands

- checkout
  - Retrieving a file from the repository
- commit
  - Putting changes back into repository
- update
  - Refresh the local or working copy copy with any changes since checkout

- http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf

## Conflict Detection and Management

- On check in, official repository copy is compared with new copy
  - Check version number
    - If repository and working copy versions are the same, accept the changes
    - If repository version is newer, reject commit operation.
- Need to update working copy before check in
  - Includes a synchronization step to merge changes from two files (new repository version and modified file from working copy)

## Conflict Detection and Management

- Merge algorithm
  - Line by line comparison
  - Changes to different lines are OK
  - Changes to same lines labeled as a conflict
    - Both versions written to the working copy copy
    - Choose which line by editing manually
  - No guarantee of code correctness after merge
- On a successful check in, save new version of files with a set of backwards references to changes
- Can apply detection selectively
  - Binary files only use version numbers or timestamps
  - No conflict detection applied to files that are ignored

## subversion Commands

- checkout
  - Retrieving a file from the repository
- commit
  - Putting changes back into repository
- update
  - Refresh the local or working copy copy with any changes since checkout

- http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf

## Conflict Detection and Management

- On check in, official repository copy is compared with new copy
  - Check version number
    - If repository and working copy versions are the same, accept the changes
    - If repository version is newer, reject commit operation.
- Need to update working copy before check in
  - Includes a synchronization step to merge changes from two files (new repository version and modified file from working copy)

## Updates

- Update command is applied to the directory
- Three possible outcomes for each file
  - U - updated, repository version newer than working copy
  - G - merged, changes don't overlap
  - C - conflict, you need to resolve
- After resolving, tell subversion client that you have done so