Software Engineering Tools
Course Summary

Created by Tali Gutman,
According to lectures by Zvi Gutterman

Written in OpenOffice Writer™

I don't promise anything about this summary, it's been adapted from lecture slides and wikipedia.

Introduction

Syllabus:

- Logging
- Source Control Management
- Build Tools
- Documentation Tools
- Quality Assurance
- Reverse Engineering
- Legal Summary

What are we looking for?

- Software development shortcuts
- Stability
- Support
- Known standards
- Easy integration with other products
  - e.g. IDE plug in, web interface
- Simple
- Scale well
- Supported platforms
- Commonly used
- Open/close source

Reading List

- Will be mentioned as we go along

Logging Tools

Logging - Definition:

- Low-tech method for debugging applications
- Can also be viewed as an auditing tool [meaning - tool for examination, verification etc...]

Basic features of any logging library are:

- Control over which logging statements are enabled/disabled
- Manage output destinations and output formats

>> Central issue is the categorization of logging statements

[From Wikipedia]
In computerized data logging, a computer program may automatically record events in a certain scope in order to provide an audit trail that can be used to diagnose problems.

Examples of physical systems which have logging subsystems include process control systems, and the black box recorders installed in aircraft.

Many operating systems and multitudinous computer programs include some form of logging subsystem. Some operating systems provide a syslog service (described in RFC 3164), which allows the filtering and recording of log messages to be performed by a separate dedicated subsystem, rather than placing the onus on each application to provide its own ad hoc logging system.

In many cases, the logs are esoteric and hard to understand; they need to be subjected to log analysis in order to make sense of them.

Other servers use a splunk to parse log files in order to facilitate troubleshooting; this approach may yield correlations between seemingly-unrelated events on different servers. Other enterprise class solutions such as those from LogLogic collect log data files in volume and make them available for reporting and real-time analysis.
[End of Wikipedia bit]

Logger – Definition:

- Named entities that follow the hierarchical naming rule.
  [example: A logger named 'com' is the direct parent of a logger named 'com.foo' and an ancestor of a logger named 'com.foo.bar']
  (Irrelevant note: Foobar, adaptation of FUBAR, stands for "Fucked Up Beyond All Recognition. Thought you'd like to know)

Log4j - A logging package for the Java Language (Ceki Gülcü wrote this logger)

Loggers, Appenders and Layouts

Log4j has three main components: loggers, appenders and layouts. These three types of components work together to enable developers to log messages according to message type and level, and to control at runtime how these messages are formatted and where they are reported.

Log4j Levels:

- Loggers may be assigned levels. The set of possible levels, that is DEBUG, INFO, WARN, ERROR and FATAL are defined in the org.apache.log4j.Level class, which I'm adding here:

[From class definition]
The DEBUG Level designates fine-grained informational events that are most useful to debug an application.
The ERROR level designates error events that might still allow the application to continue running.
The FATAL level designates very severe error events that will presumably lead the application to abort.
The INFO level designates informational messages that highlight the progress of the application at coarse-grained level.
The WARN level designates potentially harmful situations.
[End of from class bit]

- If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level.
- A logging request is said to be enabled if its level is higher than or equal to the level of its logger. Otherwise, the request is said to be disabled. A logger without an assigned level will inherit one from the hierarchy. This rule is summarized below.

Basic Selection Rule

A log request of level p in a logger with (either assigned or inherited, whichever is appropriate) level q, is enabled if p >= q.

This rule is at the heart of log4j. It assumes that levels are ordered. For the standard levels, we have **DEBUG < INFO < WARN < ERROR < FATAL.**

Log4j Appenders

- Output targets are implemented by appenders. Appenders exist for java.io.Writer, the console, files, Unix syslog, JMS [Java Message Service, it's a middleware for inter-process communications in Java. My deepest apologies to those who did not take OS yet, and my even deeper ones to those who did.], NT Event Log, SMTP and more.
- More than one appender can be attached to a logger.
- Appenders follow the logger hierarchy in an additive fashion.
  For example, if a console appender is added to the root logger, then all enabled logging requests will at least print on the console. If in addition a file appender is added to a logger, say C, then enabled logging requests for C and C's children will print on a file and on the console. It is possible to override this default behavior so that appender accumulation is no longer additive by setting the additivity flag to false.

The rules governing appender additivity are summarized below.

Appender Additivity

The output of a log statement of logger C will go to all the appenders in C and its ancestors. This is the meaning of the term "appender additivity".

However, if an ancestor of logger C, say P, has the additivity flag set to false, then C's output will be directed to all the appenders in C and it's ancestors up to and including P but not the appenders in any of the ancestors of P.

Loggers have their additivity flag set to true by default.

Layouts & ObjectRenderers

- Users can customize the output of an appender by associating with it a Layout. The layout is responsible for formatting the logging request according to the user's wishes, whereas an appender takes care of sending the formatted output to its destination. The PatternLayout, part of the standard log4j distribution, lets the user specify the output format according to conversion patterns similar to the C language printf function.
- Logging output can be formatted also in HTML or XML among other formats.
- It is also possible to format the output by message object type by registering an appropriate ObjectRenderer for a given type.
  [Just as importantly, log4j will render the content of the log message according to user specified criteria. For example, if you frequently need to log Oranges, an object type used in your current project, then you can register an OrangeRenderer that will be invoked whenever an orange needs to be logged.

Object rendering follows the class hierarchy. For example, assuming oranges are fruits, if you register an FruitRenderer, all fruits including oranges will be rendered by the FruitRenderer, unless of course you registered an orange specific OrangeRenderer.]

Configuration

- Log4j allows both programmatical and script-based configuration
- Configuration scripts can be expressed as key=value pairs or in XML format.
- Meaning: You can configure via code, or you can read *external* configuration files.

Logging performance:

- Performance when logging is turned off is extremely fast but incurs the hidden cost of parameter construction, regardless of whether the message will be logged or not.
  To avoid the parameter construction cost write:

```
if(logger.isDebugEnabled() {
  logger.debug("Entry number: " + i + " is " + String.valueOf(entry[i]));
}
```

- Performance of logging when logging is turned on is determined by the cost of walking the logger hierarchy. Typical cost of a hierarchy walk is in the range 5 to 15 microseconds.
- Actual logging is in order of 100 to 300 microseconds.

Conclusion

- Log4j is a popular logging package written in Java
- One of its distinctive features is the notion of inheritance in loggers.
- Log statements can be completely managed using configuration files. They can be selectively enabled or disabled and sent to different output targets in user chosen formats.
- Log statements can remain in shipped code without incurring a heavy performance cost.

Source Control Management

- Requirements
- Vocabulary
- Project organization
- Versioning
- Tools
  - RCS, CVS
  - Source Safe
  - Clear Case
  - Subversion
  - Additional tools

Requirements

- Software is made of many files
  - Source files, documentation, binaries, graphics, etc.
  - Complex directory tree
  - Some files are dynamically generated during the build process

➔ Which files do we store? In what structure?

- Many developers
  - Need the same files
  - Privileges
  - Offline development
  - Mistakes

- Multiple products may be base on the same source code

- Distributed development sites
  - Low communication bandwidth

- Archive and logging
  - When a function was added
  - Which tools are being used (like Log4j)
  - Management tool
    - Great for summary information.
    - Beware of abuse!

Vocabulary

http://en.wikipedia.org/wiki/Revision_control#Common_vocabulary
http://producingoss.com/html-chunk/vc.html

- Repository
  - The **repository** is where the file data is stored, often on a server. Sometimes also called a **depot.**
  - A database in which changes are stored. Some version control systems are centralized: there is a single, master repository, which stores all changes to the project. Others are decentralized: each developer has his own repository, and changes can be swapped back and forth between repositories arbitrarily. The version control system keeps track of dependencies between changes, and when it's time to make a release, a particular set of changes is approved for that release. The question of whether centralized or decentralized is better is one of the enduring holy wars of software development; try not to fall into the trap of arguing about it on your project lists.

- Working copy
  - The **working copy** is the local copy of files from a repository, at a specific time or revision. All work done to the files in a repository is done on a working copy, hence the name.
  - A developer's private directory tree containing the project's source code files, and possibly its web pages or other documents. A working copy also contains a little bit of metadata managed by the version control system, telling the working copy what repository it comes from, what "revisions" (see below) of the files are present, etc. Generally, each developer has his own working copy, in which he makes and tests changes, and from which he commits.
- Legal copy
  - To my best understanding – the opposite of an illegal copy. i.e. Software you actually paid for (in closed source).
- Check-out
  - A **check-out** (or **checkout** or **co**) creates a local working copy from the repository. Either a revision is specified, or the latest is used.
  - The process of obtaining a copy of the project from a repository. A checkout usually produces a directory tree called a "working copy" (see below), from which changes may be committed back to the original repository. In some decentralized version control systems, each working copy is itself a repository, and changes can be pushed out to (or pulled into) any repository that's willing to accept them.
- Check-in, Commit
  - A **commit** (or, more rarely, **install**, **submit**, **check-in** or **ci**) occurs when a copy of the changes made to the working copy is made to the repository.
  - To make a change to the project; more formally, to store a change in the version control database in such a way that it can be incorporated into future releases of the project. "Commit" can be used as a verb or a noun. As a noun, it is essentially synonymous with "change". For example: "I just committed a fix for the server crash bug people have been reporting on Mac OS X. Jay, could you please review the commit and check that I'm not misusing the allocator there?"
- Version
  - **Version** is a state of an object or concept that varies from its previous state or condition. The term "**version**" is usually used in the context of computer software, in which the version of the software product changes with each modification in the software.
  - The word *version* is sometimes used as a synonym for "revision", but I will not use it that way in this book, because it is too easily confused with "version" in the sense of a version of a piece of software —that is, the release or edition number, as in "Version 1.0". However, since the phrase "version control" is already standard, I will continue to use it as a synonym for "revision control" and "change control".
- Lock
  - A way to declare an exclusive intent to change a particular file or directory. For example, "I can't commit any changes to the web pages right now. It seems Alfred has them all locked while he fixes their background images." Not all version control systems even offer the ability to lock, and of those that do, not all require the locking feature to be used. This is because parallel, simultaneous development is the norm, and locking people out of files is (usually) contrary to this ideal.
    Version control systems that require locking to make commits are said to use the *lock-modify-unlock* model. Those that do not are said to use the *copy-modify-merge* model. An excellent in-depth explanation and comparison of the two models may be found at http://svnbook.red-bean.com/svnbook-1.0/ch02s02.html. In general, the copy-modify-merge model is better for open source development, and all the version control systems discussed in this book support that model.
- Trunk, Branch
  - A copy of the project, under version control but isolated, so that changes made to the branch don't affect the rest of the project, and vice versa, except when changes are deliberately "merged" from one side to the other (see below). Branches are also known as "lines of development". Even when a project has no explicit branches, development is still considered to be happening on the "main branch", also known as the "main line" or "*trunk*".
    Branches offer a way to isolate different lines of development from each other. For example, a branch can be used for experimental development that would be too destabilizing for the main trunk. Or conversely, a branch can be used as a place to stabilize a new release. During the release process, regular development would continue uninterrupted in the main branch of the repository;

meanwhile, on the release branch, no changes are allowed except those approved by the release managers. This way, making a release needn't interfere with ongoing development work. See the section called "Use branches to avoid bottlenecks" later in this chapter for a more detailed discussion of branching.

- Tag, Label
  - A label for a particular collection of files at specified revisions. Tags are usually used to preserve interesting snapshots of the project. For example, a tag is usually made for each public release, so that one can obtain, directly from the version control system, the exact set of files/revisions comprising that release. Common tag names are things like Release_1_0, Delivery_00456, etc.
- Merge (aka Port)
  - A **merge** or **integration** brings together (merges) concurrent changes into a unified revision.
  - To move a change from one branch to another. This includes merging from the main trunk to some other branch, or vice versa. In fact, those are the most common kinds of merges; it is rare to port a change between two non-main branches. See the section called "Singularity of information" for more about this kind of merging.
    "Merge" has a second, related meaning: it is what the version control system does when it sees that two people have changed the same file but in non-overlapping ways. Since the two changes do not interfere with each other, when one of the people updates their copy of the file (already containing their own changes), the other person's changes will be automatically merged in. This is very common, especially on projects where multiple people are hacking on the same code. When two different changes *do* overlap, the result is a "conflict".
- Conflict (Not from slides)
  - What happens when two people try to make different changes to the same place in the code. All version control systems automatically detect conflicts, and notify at least one of the humans involved that their changes conflict with someone else's. It is then up to that human to *resolve* the conflict, and to communicate that resolution to the version control system.
- Metadata
  - Log, author, date..

Project Organization

- First decide on your directory structure
  - Source files
  - Headers
  - Documentation
  - Build logs
  - QA
    - Test suits
    - Test results
  - Binaries – which do we need?
    - Everything that goes to a version – yes.
    - Everything else – no.
    - Usually:
      - Executables + DLL's [including debug versions]
      - No Object files.
    - One day we will need them and the overhead is not so big
    - Redundancy is fine (better than the other way around...)
  - Plan in advance!
    - Changing the structure will be expensive
    - Usually a cross R&D process

Versioning

- Global product or branch number tag
- Used to mark software maturity
  - Good practice
- Do not mix with the marketing numbering (i.e. This refers to inner versioning)
- Select meaningful numbering system
- Add indication for Alpha and Beta versions [Wiki, Development Stage]
  - Alpha version:
    - The **alpha** version of a product still awaits full debugging or full implementation of all its functionality, but satisfies a majority of the software requirements. It often lacks features promised in the final release, but demonstrates the feasibility and basic structure of the software.
      The alpha build of the software is the first build delivered to the software testers.
      In the first phase of alpha testing, developers test the software using white box techniques. Additional inspection is then performed using black box or grey box techniques. This is usually done by another dedicated testing team sometimes concurrently. Moving to black box testing is often known as the second stage of alpha testing.
      The name is derived from alpha, the first letter in the Greek alphabet.
  - Beta version:
    - A **beta** version or **beta release** usually represents the first version of a computer program that implements all features in the initial software requirements specification. It is likely to be unstable but useful for internal demonstrations and previews to select customers, but not yet ready for release. Some developers refer to this stage as a *preview*, as a *technical preview* (TP) or as an *early access*.
      Often this stage begins when the developers announce a feature freeze on the product, indicating that no more features requirements will be accepted for this version of the product. Only software issues, or *bugs* and unimplemented features will be addressed.
      Beta versions stand at an intermediate step in the full development cycle. Developers release them to a group of **beta testers** (sometimes the general public) for a user test. The testers report any bugs that they found and sometimes minor features they would like to see in the final version.

RCS – Revision Control System

- Written by Walter Tichy, Purdue University [early 1980's]
- www.cs.purdue.edu/homes/trinkle/RCS
- Included in Unix, Linux, Cygwin by default.
- Open Source
- File perspective using Unix diff and hence bad with binaries.

CVS – Concurrent Versions System

- Open source
- www.cvshome.org
- Use RCS for file level
- Very stable
- Multi OS: Win, Unix, Linux, Mac, VMS
- Default control in command line
- Probably with the largest installation base today
- Default tool for:
  - Sourceforge
  - Eclipse
- Large number of plug ins

- Default: Non-locking. (i.e. Copy-modify-merge)

Visual SourceSafe (VSS)

- Microsoft, and thus used in many projects by default.
- Binary, proprietary *[Owned by a private individual or corporation under a trademark or patent]* repository structure.
- Good integration into Visual Studio (surprise, surprise...)
- Windows platform only
- Bad network performance
- GUI based interface
- Default – locking. (i.e. Lock-modify-unlock)
- Not in use to develop Windows...
- Vault – VSS like replacement.
  - VSS minus its problems
  - www.sourcegear.com/vault/index.html

Team System

- Microsoft new toolset
  - Integrated in the new VS (2005)
- http://msdn.microsoft.com/vstudio/teamsystem/
- Now in beta (May 2005)
  - To be released in November 2005
- Teamsystem FAQ [Or why do some developers hate microsoft?]:
  - **Q:** Is this the tool that Microsoft uses internally?
  - **A:** Not yet. However, once Visual Studio 2005 Team System ships, the Developer Tools Division at Microsoft will be using the source code control system full-time. This is one of our core tenets for change management: We will build the product we use internally.

Clear Case

- By IBM (Rational)
- http://www-306.ibm.com/software/awdtools/clearcase
- OS support: Windows, Linux, Mac, Mainframe
- Heavy-weight
  - Large installation (RAM, HD)
  - Usually arrives with a sys admin...
- Now in three flavors
  - Light (LT), "regular", Multi-site
- Leading the high-end market with share of 30-40%
- Support for offline work
  - "View" in rational terms
  - Including virtual local file system.

Subversion

- http://subversion.tigris.org
- Young project
  - Goal – CVS replacement
  - Development started in 2000

- Based on Apache web server
  - Improved network efficiency
- Better support for binary files
- plug ins
  - http://tortoisesvn.tigris.org

Additional SCM tools

[http://en.wikipedia.org/wiki/Comparison_of_revision_control_software]

- SCCS – Source control system.
  - RCS predecessor
  - Maintained at SourceForge
  - Was the first source code revision control system.
  - Today, generally considered obsolete.
  - uses a storage technique called interleaved deltas (or the weave). This storage technique is now considered by many revision control system developers as key to some advanced merging techniques
- AllFusion Harvest
  - All-in-one by CA
  - http://www3.ca.com/Solutions/Product.asp?ID=255
  - Features on glance
    - Process-driven, integrated change and configuration management
    - Powerful change and defect tracking
    - Straightforward inventory management
    - Easy version control and release management
    - Automated build management
- Serena Version Manager
  - PCVS ancestor
  - Copy-Modify-Merge
  - http://www.serena.com/Products/professional/vm/home.asp
- Perforce (http://www.perforce.com)
  - http://en.wikipedia.org/wiki/Perforce
  - Copy-Modify-Merge
- Bitkeeper (http://www.bitkeeper.com)
  - Former Linux SCM
  - Copy-Modify-Merge
- StarTeam
  - Borland
  - Targeted for high-end
  - http://www.borland.com/starteam/
- Git
  - "The stupid content tracker"
  - New Linux version system
  - Improved interface by cogito
    - http://www.kernel.org/pub/software/scm/cogito/README
  - Use only if you must

Build Tools

- Requirements
- Vocabulary
- Build tools
  - Make
  - Ant
- Continuos Build Process
  - Cruise Control
  - AntHill

Requirements

- Input: Source files
  - Code, images, encryption keys,...
- Output: Product
  - Usually binary made of sources
  - May also include
    - Documentation files
    - Installation tools
  - Completely automatic
  - Email notifications
  - SCM integration (Reading sources)
  - Timely build (Daily)
  - Integrate QA to the process

Vocabulary

- Action (=task) - Any activity – compilation, file copy, etc.
- Target
  - Action results
  - Binary for example
- Dependencies
  - Relation definition. Target should be refreshed according to it's sources.
- Timestamps
  - Refreshing is done on a time-stamp base only.

Solutions

- Script
- GUI tools (i.e. Visual Studio)
- Tools such as:
  - Make
  - Ant

- Make
  - 1977 by Stuart L. Feldman (Also the author of Fortran 77)
  - Open Source
  - Many platforms
  - Many variations (e.g. Gmake [GNU Make])

- Utility named *Make*
- Rules are defined in *Makefile*
- Running Make
  - Will not rebuild anything that's up to date
  - Make will stop on any action error
  - Error in dependency will stop Make
  - Make is an interperter
- Dependency Files
  - Most C compilers generate a *Make* rule when invoked using "gcc -M file.c"
  - Common practice: Define an implicit rule "%.d:%c" that runs "gcc -M file.c>file.d".
  - include *.d – includes other makefiles in this one.
  - Smart enough to rebuild the *.d files first so it knows what to rebuild.
- Limitations
  - Cross platform problems
  - Difficult syntax
  - Lack of support for distributed building
- Make extensions
  - Jam
    - http://www.perforce.com/jam/jam.html
    - Make + Cleaner Syntax + Implicit dependency rules for C/C++
  - Visual Studio
    - nmake.exe
    - Dialect of Make

- Compiler Cache
  - C/C++ compilation cache
  - Acts as pre-processor
  - Performance up to 10 times faster
  - ccache [http://ccache.samba.org/]
  - compilercache [http://www.erikyyy.de/compilercache]

- XOREAX
  - Israeli build company
  - Distributed tool to improve performance
  - Plugin for Visual Studio
  - http://www.xoreax.com/

- Ant
  - http://ant.apache.org
  - Open source
  - Started in 2000 [James. D Davidson]
  - Main focus is Java
    - Cross platform task
    - Made to build Jakarta
  - Written in Java
  - "Makefile" is written in XML
    - Default: build.xml
  - .net version called *Nant* [http://nant.sf.net]

- Installing Ant
  - Get the binary release and set ANT_HOME and JAVA_HOME environment variables. Put the ant bin directory on the PATH environment variable.
- Main Ant Tasks
  - Javac
  - Javadoc
  - jar
  - Mkdir
  - Copy
  - Delete
  - Junit (tests)
  - You can also write your own tasks.
  - Different Task Categories
    - Archive Tasks
    - Audit/Coverage Tasks
    - Compile Tasks
    - Deployment Tasks
    - Documentation Tasks
    - EJB Tasks (Enterprise JavaBeans is the server-side component architecture for Java Platform, Enterprise Edition)
    - Execution Tasks
    - File Tasks
    - Java2 Extensions Tasks
    - Logging Tasks
    - Mail Tasks
    - Miscellaneous Tasks
    - .NET Tasks
    - Pre-process Tasks
    - Property Tasks
    - Remote Tasks
    - SCM Tasks
    - Testing Tasks
    - Visual Age for Java Tasks (Visual Age for Java is a great Java IDE, but it lacks decent build support; for creating deliveries. On the other hand, Ant supports the build process very good, but is (at least at the moment) command line based.)

Continuous Integration

- Have a build iteration on each SCM commit:
  - Get latest source version
  - Build
  - Run tests
  - Notify results to developers
- Everything is done automatic once a programmer checks in a file.
- CI Tools
  - AntHill [http://www.urbancode.com/]
  - CruiseControl [http://cruisecontrol.sourceforge.net/]

Documentation Tools

- Doxygen
  - Documentation system
  - www.doxygen.org
  - Goals
    - Code manual
      - similar to javadoc, but much more extended
    - Reverse engineering
      - Extract code structure
    - Mainly for C++
  - Developed by Dimitri Van Heesch
  - Open source since 1997
  - Simple use
    - Reverse engineering [Can extract code structure from un-documented code]
  - Extended Use
    - Add documentation directives in your code.
    - Syntax similar to javadoc and output is HTML, adding '!' to C/C++ comments. i.e. //!comment or /*! comment */, when latter is considered an elaboration of the former.
    - See example in slides.
    - Also supports embedded LaTex (\f[formula \f])

Quality Assurance

- Goal
- Introduction
- Vocabulary
- White Box
- Black Box
- Static and Dynamic testing

- Question: Who has the highest Salary in Israel public service in 2001?
- Answer: The test pilot of IAI! Why? Because there's always another BUG.

Goal – QA

- The process of executing a program (or part of it) with the intention of finding errors. [Myers, "The Art of Software Testing" 1979]
- Any activity aimed at evaluating attribute of a program and determining that it meets its required results. [Hetzel,"The Complete Guide to Software Testing" 1988]

Introduction

- 30% - 90% of software life cycle
  - HUJI vs. Air Craft Industry
- Usually not done by professionals
- Not an exact science (can be impossible)
- Complete coverage is impossible!
  - Number of option is exponential
- Fixes introduce new bugs

- OS & Hardware dependent
- Programming Language dependent
  - C++ Vs. Java
- Never Ends (client as a tester)
  - There is always another bug
- Marketing vs. Testing
  - 1980's – PC
  - 1990's – Internet hype
- Requires writing additional code (which can introduce bugs)
- Automatic tools for testing exists
  - Mainly for known fields
    - Web servers
  - Parasoft
  - Mercury Interactive

Vocabulary

- Bug, Fault, Error, Defect, Errata, Feature
- Fix, Patch, Service Pack, Update, Upgrade
- Versions: Alpha, Beta
  - Release Candidate
    - The term **release candidate** refers to a final product, ready to release unless fatal bugs emerge. In this stage, the product features all designed functionalities and no known showstopper class bugs. At this phase the product is usually *code complete*. ...
      A release is called *code complete* when the development team agrees that no entirely new source code will be added to this release. There may still be source code changes to fix defects. There may still be changes to documentation and data files, and to the code for test cases or utilities. New code may be added in a future release. [http://en.wikipedia.org/wiki/Development_stage]
  - 1.1.x VS. 1.2.x

Event	Version
First released version	1.0
First revision	1.1
First bug fix to the first revision	1.1.1
First major revision or rewrite	2.0

    (So in our case, the difference between them is the revision number)
- Regression/Functional/White Box/Black Box/ Unit/ System/ Integration Testing

White Box – Unit Testing

- Testing the Implementation Logic
- Usually done by the programmer
  - good/bad?
- Tests the source code
- Example: Implementing a Sorting Method
  - Merge Sort Vs. Bubble Sort

Black Box – Functional Testing

- Use Cases and Requirements based
- Interface testing
- Input-output testing
- Use Valid and Invalid inputs
- Use "real" samples
- Unaware of the used algorithm
- Example: Sorting
  - Use complete enumeration for small examples
    - 10 elements (10!)
    - Should we test again for 11 elements?
  - Check for equal values (2,2,2,5,3,4,7)
  - Which input ranges are valid? (use extremes)
- Random Testing (why?)
  - Because real life input is really unexpected and thus random.

Testing affects results

- Test Programs
- Another Software is using the same resources
- It is not the same code
  - Debug vs. Release
    - For example, counters are initialized to zero (gdb)
    - assert(x=3)
      - (Implies that we sometimes [erronously] write *actual* code in debug context)

Static Testing

- Static [http://en.wikipedia.org/wiki/Static_testing]
  - A form of software testing where the software isn't actually used.
  - Compiler
    - C++: -Wall, -pedantic, -ansi
    - The type checking by a C compiler is an example of static verification.
  - Static Checker
    - -e.g. - Lint (http://lclint.cs.virginia.edu)
  - Human code review
  - Formal verification

Dynamic Testing

- Runtime monitors
  [Performing the test with the application running.]
  - Memory management
    - Debugger
    - Unique tools – Purify[http://en.wikipedia.org/wiki/Purify] /Bounds Checker [like memcheck]
  - Timing
    - Quantify [counters are inserted at the object code level to time every assembly code sequence, providing performance analysis data down to the source line on every portion of an application's code.]
    - Microsoft

- Coverage
  - Pure coverage
    - Using PureCoverage, you can:
      - Identify the portions of your application that your tests have not exercised
      - Accumulate coverage data over multiple runs and multiple builds
      - Merge data from different programs sharing common source code
      - Work closely with Purify to make sure that Purify finds errors throughout your *entire* application
      - Automatically generate a wide variety of useful reports
      - Access the coverage data so you can write your own reports

      PureCoverage provides the information you need to identify gaps in testing quickly, saving precious time and effort.

Testing Metrics

- Number of bugs
  - Day/Developer/Team/Program
- Define severity levels
  - No go! (showstopper)
  - ...
  - Feature
- Regression Tools (re-approving the product)
- Acceptance Testing [http://en.wikipedia.org/wiki/Acceptance_test]
- Poor man solution:
  - Time/Budget

Test-First Design (TfD)

- Test-first design is one of the mandatory practices of Extreme Programming (XP)
- Programmers must not write any production code until they have first written a unit test.

Junit

- Homepage – www.junit.org
- A testing framework for Java
- Developed by Kent Beck and Erich Gamma
- Versions exist for other languages
  - e.g. Nunit for .NET
- Junit has been 'integrated' into some IDEs

Rational Purify (IBM)

- A tool for locating runtime errors in a C/C++ program.
- Purify can find:
  - Array bounds errors
  - Accesses through dangling pointers
  - Uninitialized memory reads

    - Memory allocation errors
    - Memory leaks
- Works for Windows & Linux & Unix
- Inside Purify
    - Instruments a program by adding protection instructions around every memory action.
    - This is a static translation
    - When program is executed a viewer will be created to display errors as they happen.
    - Purify can also run standalone with any executable.
    - Makefile:

```
program: $(OBJS)
    purify [-option ...] $(CC) $(CFLAGS) -o| program $(OBJS) $(LIBS)
```

    - Also support dynamic mode

Valgrind

- www.valgrind.org
- open source
- Linux
- Dynamic binary translation
- *Memcheck*

  Memcheck detects memory-management problems, and is aimed primarily at C and C++ programs. When a program is run under Memcheck's supervision, all reads and writes of memory are checked, and calls to malloc/new/free/delete are intercepted. As a result, Memcheck can detect if your program:
    - Accesses memory it shouldn't (areas not yet allocated, areas that have been freed, areas past the end of heap blocks, inaccessible areas of the stack).
    - Uses uninitialised values in dangerous ways.
    - Leaks memory – pointers to malloc'd blocks are lost forever.
    - Passes uninitialized and/or unaddresible memory to a system call
    - Mismatched use of malloc/new/new [] vs. free/delete/delete []
    - Does bad frees of heap blocks (double frees, mismatched frees).
    - Passes overlapping source and destination memory blocks to memcpy() and related functions.

    Memcheck reports these errors as soon as they occur, giving the source line number at which it occurred, and also a stack trace of the functions called to reach that line. Memcheck tracks addressability at the byte-level, and initialisation of values at the bit-level. As a result, it can detect the use of single uninitialised bits, and does not report spurious errors on bitfield operations. Memcheck runs programs about 10--30x slower than normal.

Reverse Engineering

- Goals
- Complexity?
- Tools
  - IDA
  - Softice
  - LXR
- Copy Protection

Reverse Engineering – why?

- Bad or no documentation
- Competition
- Interfacing your software with others
  - e.g. You want to build a plugin for ICQ
- Security studies
  - Vulnerability assessments

Complex Task

- Compilation is a one way function
- Considered a good encryption
- Different levels of optimizations
- Not a one-to-one mapping
- No standard code translation
- Usually your building blocks are in assembly language
- Simple case?
  - Virtual Machines
    - MSIL (Microsoft Intermediate code)
    - Bytecode (Java)
  - Both contain much more information than assembly code
  - Debug versions (include more information and thus easier to reverse-engineer)

Tools

- Source vs. Assembler Analysis
- Static Analysis
  - IDA
  - LXR
- Dynamic Analysis
  - Debugger
  - Softice

IDA

- Static, interactive dis-assembler
- http://www.datarescue.com/idabase
- Windows and Linux
- Multi-processor
  - Probably any CPU you may need
  - Including Intel, RISC, .NET, JVM

- Can output a flow analysis [What happens if true, what happens if false...]

Softice

- http://www.compuware.com/products/driverstudio/softice.htm
- By Compuware
- Windows only
- Kernel level debugger
  - Allows breakpoint at ANY assembler line
- http://en.wikipedia.org/wiki/SoftICE
- Crucially, it is designed to run underneath Windows such that the operating system is unaware of its presence. Unlike an application debugger SoftICE is capable of suspending all operations in Windows when instructed. For driver debugging this is critical due to how hardware is accessed and the kernel of the operating system functions.
- Originally developed by NuMega and purchased by compuware.
- As of April 3rd, 2006 the DriverStudio product family [which includes softICE] has been discontinued because of "a variety of technical and business issues as well as general market conditions". Maintenance support will be offered through March 31, 2007.

LXR

- *LXR Cross Referencer*, usually known as *LXR*, is a general purpose source code indexer and cross-referencer that provides web-based browsing of source code with links to the definition and usage of any identifier. Supports multiple languages.
- Source code indexer
- Cross-referencer
- web based
- http://sourceforge.net/projects/lxr
- Example: Linux kernel source code
  - http://lxr.linux.no

Copy Protection[http://en.wikipedia.org/wiki/Copy_protection]

- Mission impossible
- How to delay the reverse engineering
- Code encryption
- Registration
- Licensing [Serial numbers, registration key...]
- Anti-tampering
- Don't waste too much time on it...

Obfuscators [http://en.wikipedia.org/wiki/Obfuscating_software]

- Trying to make de-compiling harder
- Mostly is source code but sometimes also assembly code
- Goal
  - Hard to understand
  - Same functionality
  - Same performance
  - No new bugs...

Intellectual Property (IP)

- License
  - When developing your own project
  - When using a 3rd party product
  - Examples
  - Blackduck software
  - EULAyzer
- Software Patents
- Disclaimer: Zvi is not a lawyer – get professional help when needed.
- If you go to IE->Help->About, you see a list of legal delegations for every possible thing they even considered using :)

Software Licensing

- When will you be dealing with licensing?
  - When you want to sell/publish/deliver your software
  - When you are using someone else's code [binary or source]
    - Otherwise you may be violating someone's license

Related Business Descisions

- Revenue model
  - Service vs. off-the-shelf
  - Who is your client
    - e.g. Who's Google's clients?
  - Software leasing
    - Software leasing is a sales and marketing tool which provides users with a lease financing alternative to an upfront cash payment. Leasing often results in an increase in revenues, a decrease in sales time, and an increase in user satisfaction levels. [http://www.lpilease.com/benswl.htm]
    - IBM Mainframe
  - Per seat, per machine, per cpu, per operations [http://idlastro.gsfc.nasa.gov/idl_html_help/Licensing_Methods.html]
    - Floating [A floating (not node-locked) license is a license which is generated by a license server and is not locked to a particular PC. This enables a team of developers to share a pool of SN Systems licenses more efficiently.]
    - Node locked [Ties a single software application to a single machine.]
  - Royalties [http://en.wikipedia.org/wiki/Royalties]
    - The *royalty* is typically a sum of money to be paid to the owner or Licensor of Intellectual Property IP Rights for the benefits derived, or sought to be derived by the user (the Licensee) through the exercise of such rights. Royalty (the word is usually used in the plural - royalties) may be paid for the use of copyright, patent, registered design, knowhow or trademark or a combination of them.
  - Licensing [http://en.wikipedia.org/wiki/Licensing#Licensing]
  - Exclusiveness [http://en.wikipedia.org/wiki/Intellectual_property#Exclusive_rights]

Additional Decisions

- Credit
  - Do I get the credit vs. White label

    - http://en.wikipedia.org/wiki/White_label_product
- Source availability
- Permission to redistribute
- Permission to modify the source

Open Source

- Initiative [www.opensource.org]

- Guidelines
    - Free redistribution of software
    - Distributions with source code
    - Permission for derived works under the same license
    - Distributing modified code or patched files allowed
    - No discrimination against persons or groups
    - ...

- Very trendy
    - Linux
    - Apache web server
    - Java Libraries, PHP, Perl, Python
    - MySQL, Postgress
    - Eclipse
    - Emacs, VI
    - OpenOffice
    - Jboss
    - CVS, Subversion
    - SourceForge

- Good and Bad

Good	Bad
Free (cost, change)	No one to blame
Ability to customize	Lack of support (no contract)
Easy to asses (quality, security)	Project goal may change
24/7 support – (Community?)	Licensing issues
Cool	Lack of central management
Vendor independent	

- Main Use:
    - GNU General Public License (GPL)
    - Mozilla
    - BSD

- Additional examples can be found in http://www.opensource.org/licenses

GPL V2 [1991]

- http://www.gnu.org/copyleft/gpl.html
- You can copy, modify, distribute the source

- Your resulted product must be GPL as well (Viral)
- The critical point – linking or not
  - If you link – viral GPL
  - Not link (even on the same media) - ok
- No sharing needed if not redistributed
  - e.g. Your own web server
- Patents must grant use with no fee
- GPL v3 to be released later in 2006
- Gnu Lesser General Public License (LGPL) is a weaker license which supports linking with non GPL software
- No warranty, ofcourse.

Richard Stallman definition of GPL [http://en.wikipedia.org/wiki/Richard_Stallman] The founder of GNU

- Freedom zero is the freedom to run the program as you wish, for any purpose
- Freedom one is the freedom to study the source code and change it to make it do what you wish
- Freedom two is the freedom to help your neighbor, that's the freedom to make copies and distribute them to others when you wish.
- Freedom three is the freedom to help your community: The freedom to publish or distribute modified versions when you wish.
- *On copyleft:* "Copyleft means that the license actively defends the freedom of all users and what this means is: we don't just stop with giving you these four freedoms but we ensure that every user that gets a copy of the program gets the same four freedoms in the same way. "

BSD

- Berkley Source Distribution
- Started with a license for Unix dialect
- http://www.opensource.org/licenses/bsd-license.php
- Not viral
- Permission to modify and redistribute and use as long as you mention the original source.
- No warranty.

Blackduck Software

- http://www.blackducksoftware.com/index.html
- Automatic tool for scanning source code and finding IP licensing issues.
- Based on patterns of licenses
- Based on known packages (e.g. SF.net)
- Zvi didn't try it

EULAyzer

- http://www.javacoolsoftware.com/eulalyzer.html
- Analyze license agreements for interesting words and phrases
- Zvi tried version 1.1 (latest, June 2006) and did not get anything meaningful on:
- GPL v.2

- Microsoft outlook 2003
- Only relevant thing was that the Eulayzer found the MS license more restricted …

Software Patents

- The word "patent" originates from Latin word "patere" which means to make available for public inspection
- Motivation:
  - A limited monopoly is granted by a government to an inventor to make, use or sell an invention, for a limited term in exchange for laying the invention open for public inspection
- Go and see some examples online
  - uspto.gov
- Requirements – Patentability
  - Original works
    - If the algorithm is already published (e.g. A paper, a conference) you can not claim patent on it.
  - "Invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent"
  - Very vague definition
    - Easy case – New algorithm
      - RSA (asymmetric encryption)
      - JPEG
    - Difficult case
      - Your innovative spreadsheet product
- Patents are valid for 20 years.
- Per country, but there are certain agreements between countries.
- Cost
  - Initial cost may be small ~hundreds $
  - Next steps are very pricey ~tens of thousands $
    - Lawyers
    - Office actions (translations)
    - Filling costs per country
- You can not ignore:
  - Your own great algorithm
  - When using someone else's patent
    - Many people user other encryptions to avoid the RSA patent
- It's a great marketing tool
  - Especially if you are running a start-up
- When you get to court number of patents does matter
- Reward employees for innovation
  - In some companies employees get paid for new patents
    - May cause inflation
- Complex legal status in some cases
  - Employee working on private non-enterprise patent
  - University
    - You may not own the IP for your projects.

Additional IP issues

- Trademarks [http://en.wikipedia.org/wiki/Trademark]
- Standards:
  - If you make a standard based on your IP you can reach to a status where everybody is paying you for royalties…
  - You write standard you may conflict with your IP (the standard uses other technology for example).