Aegis is Only for Software, Isn't It?

Peter Miller
pmiller@opensource.org.au

Aegis is a Software Configuration Management system, which provides a method for managing concurrent development and peer review with strong auditability. These features are useful to more environments than the development of software. This paper presents two systems being managed with Aegis at AGSO: DNS and the Web.

Using Aegis to manage DNS provides a reliable way to add entries to the DNS tables, check the tables, and generate the reverse maps. It is even generating some NIS+ tables. The system is peer reviewed, so no "broken" changes are able to get into the system tables. The net result is a more reliable service to the users.

Using Aegis to manage the AGSO Web (http://www.agso.gov.au/) models the production of scientific papers. In the normal publication process an author writes a paper and it is then peer reviewed, the reviewers may return it with comments or approve it to the publisher. The publisher in turn may accept it for publication or return it. A similar model is available using Aegis when publishing Web pages; the publication analogy is deliberate since the work is indeed available to the public. The "build" step which is so central to producing software is used in the Web case to resolve server-side includes, to check the HTML for obvious errors, and to generate the various indices. Some Aegis reports are also used, such as the one from which the "What's New" page (http://www.agso.gov.au/-whatsnew/) is generated. The provision by Aegis of individual "sand pits" greatly facilitates concurrent development of Web pages and improves productivity. In general, staff have been happy with the Web development model in use at AGSO.

1. Aegis at a Glance

The term *Configuration Management* was coined many decades ago in the engineering disciplines. It refers to the process of managing all of the components of an engineering design. In designing a car, for example, it is important to know which version of the shock absorbers goes with which versions of the chassis and wheel arches.

A similar problems exists for the construction of software, and so the term was borrowed by software engineers to become *Software Configuration Management.* This refers to keeping track of the various source components of a software package, and how to assemble them into a working program.

There is a huge similarity between assembling a software program, and assembling a web. Source files are processed and stitched together to form the final product, much the same way program source files are compiled and linked together to form the final software product.

Aegis is a software configuration management system, and it provides controlled accesses to the source files of a project. It can support concurrent development, and enforces mandatory reviews. As Aegis is described more completely elsewhere [1, 2], this paper will not dwell too heavily on how to use Aegis, but rather how Aegis is useful in the two case studies below.

2. DNS Management

The first case study presented here is that of using Aegis to manage Domain Name Service (DNS) information. This is a relatively small system, it is reasonably well known, and if you get it wrong your network may cease to function until you get it right again. DNS itself has been described

elsewhere [3, 4] and will not be described in depth here.

The design of the Unix *named*(8), the program which serves the DNS protocol, is intriguing, mostly because of the almost-but-not-quite redundancy in most of the configurations files. This revolves around the fact that DNS has both *forward* maps which associate a domain name with an IP address, and *reverse* maps which associate an IP address with a domain name. Naturally, if these maps are not carefully synchronized, a variety of problems may result. This is traditionally solved using *make*(1) and some baroque *awk*(1) scripts. The maintainer logs in as *root*, edits the forward map, runs make, and informs named that it needs to re-read its input files, usually by running a shell script which sends a signal to the server process.

But what do you do when there is more than one authorized maintainer? How do they coordinate their activities when they are in 3 different buildings? How do they make sure the configuration files are OK *before* they bring the whole network to its knees?

None of these problems are particularly new or interesting. They are well known to software development teams the world over. At AGSO, we are using Aegis to resolve them.

2.1. Concurrent Development

Aegis has the concept of a *baseline* which is the known-good currently-working master source for a project. Aegis partitions alterations to this baseline into sets of files which must be altered simultaneously to preserve the currently-working-ness of the baseline. These sets of files may be as small as a single file, or as large as every source file in the baseline.

These alterations to the baseline are known, unoriginally, as *changes*, and each change is given a separate development directory. By developing each change in a separate directory, there is no possibility than one maintainer can accidentally blow away another's work. At worst, if they have a file in common, one will need to merge the other's work with his own. This merge is fully supported by Aegis.

The other important feature of using separate development directories is that "half finished" changes are not mistaken for valid input, should the DNS server be rebooted during the development of the change. Until a change is completed *and* has been reviewed, the baseline remains unaltered.

This entire process is performed by users logged in as themselves. Access is controlled by access control lists based on user names, no special system user accounts are required. need for users to login to a special account to work on web pages.

2.2. Validation

When constructing software, the program is built in some way. One traditional Unix approach is to delegate remembering how to do this to a program such as *make*(1). While *make*(1) can be used with Aegis, the author prefers *cook*(1) as a more capable and descriptive build tool.

The build step is used to translate the forward maps into reverse maps. This is not done with *awk*(1) for a few reasons

- The SRRF input format used by *named*(8) makes for very hard to read awk scripts if you are determined to use the full input language.
- We use include files to describe our various buildings and other internal network structure. This also reduces the number of opportunities for conflicts between changes.

As a result, we have a suite of C programs (also developed using Aegis, naturally) which perform the various translations and filters we require.

A very important side-effect of these translations is that they validate their input. In this we detect duplicate host names, invalid host names, duplicate IP addresses, invalid IP addresses, etc. When we converted our DNS procedures to use DNS, a number of problems were discovered which had been in the system for years.

Yes, these problems could have been discovered with the old root make system, but they were not discovered until we decided to us a system specifically designed to be maintained by many people, and to double check everything before it went "live" and inconvenienced staff.

2.3. Derived Data

We also use the DNS data to generate some of our NIS+ tables. Obviously, the NIS+ hosts table needs to be synchronized with the DNS tables, and this translation is relatively easy.

We also invented an "ether" type, which we filter out and don't actually give to *named*(8), to record Ethernet addresses. This information is used to derive the NIS+ ethers table. This could have been managed differently, but as it is intimately

related to the IP address management, we decided to do it here, for a sort of "one stop" network shop.

The `hinfo` records are *grep*(1)ed to generate the NIS+ netgroup table. This is used to minimize changes to */etc/hosts.equiv*, */etc/dfs/dfstab*, etc. As a new machine is added, it receives a formula `hinfo` description, which results in it being included in the appropriate net group, and thus the appropriate network permissions.

The `hinfo` records are also filtered to generate the NIS+ bootparamd map. When combined with the hosts table and the ethers table, this is all that is required to configure a workstation for the Solaris Install Server.

2.4. Review

Aegis will not allow a change to finish development until it builds cleanly. Thus, if there are any errors found during the various filters performed by the build step, a change may not end development.

Once a change ends development it is not immediately installed into the baseline. It enters a "being reviewed" state. Developers are prevented from reviewing their own work, as an obvious conflict of interests. Some other authorized review must do so. The access control lists for developers and reviewers are separate, and it is up to your individual preferences whether they overlap or not. Once a reviewer OKs the change, it is then integrated.

2.5. Integration

The integration involves making a copy of the baseline (usually with hard links, it is faster) and applying the change to this copy. The build is performed again on this copy - mostly as a quick double check, though it does not in the Web case, below.

The integrator can also serve as an editor, or a second reviewer if necessary. It answers the perennial "who will watch the watchers" question.

Aegis' notification facility, used at most transition to send email or news articles, is also used at the end of integration to notify DNS and NIS+ to reread the relevant files.

2.6. Observable Results

It was long suspected that there was a fairly high rate of change in our DNS data, but we didn't track it too closely, we already had more than enough to do. Aegis tracks this activity in a very non-intrusive manner. At present, we are averaging 3.5 DNS changes per week, a figure not previously available.

The other thing which has changed is that there are no unexplained changes. You no longer hear they cry "Who the smeg did that?" echoing down the corridor. There is never the silent anonymous untrackable duel between two administrators undoing and redoing each others changes.

By adding Ethernet numbers, we can detect when machines move between staff and between floors and change names, meaning that we can detect now-vacant IP addresses by the Ethernet address duplicate.

The most important result is stability. More is checked than ever before, and it is discovered earlier than ever before. Only changes which pass all our automated checks, and also pass the scrutiny of a human DNS boffin, are inflicted on users.

3. Web Management

The second case study presented is that of AGSO's Web. This is also managed using Aegis, though this project has many, many more sources files. The environment presented by Aegis is the same - though the terminology used is more aimed at authors than system administrators.

The method of Web development implemented using Aegis is presented as an analogue of the publications of scientific papers.

In particular, it is expected that much of the science content of our web pages will be authored by out scientists. The peer review is expected to be, and should be, performed by the scientist's science peers. Everything else - layout, formatting, copyright issues, structure, etc, are the editor's problem. Only the author may alter a change - the peer reviewer and the editor only have the power of veto. The file system permissions are exploited to ensure that there is no "back door' to this process. Even the editor (integrator) does not have write permissions to the baseline. All changes to the baseline *must* traverse the entire change process.

3.1. Build and Validation

The build step in the Web project is very little different from the build step in the DNS project. We use the build step to:

- Resolve server-side include files, so that the files fetched by external clients are complete, and the HTTP server can be as dumb as possible. We use common headers and footers in common include files to implement much of the look and feel of our web pages.
- Perform SGML checking against the HTML 2.0 DTD. The finds all of the overt HTML errors, but it cannot check spelling or style. We use *sgmls*(1) obtained from the network, and the standard `html.dtd` also obtained from the network.

It takes longer to perform this build than in the DNS case, because there are more source files and more include dependencies to check.

The build works out what to build from the source files of the project. The *cook*(1) programs asks Aegis for a list of files, and then generates a list of output files from that. In this way, there is no need to add a new file to a list in a *Makefile* of some sort.

3.2. File Templates

To add a new page to the Web, a user initiates a Web change and then creates a new file via Aegis. The new file will be created according to a preconfigured file template. This will give them the common headers and footers automatically, all that is required is to fill in the text in the middle.

So that the page may be accessed, it will also be necessary to copy some other file, again via Aegis, and embed an anchor to the new page in the copied page.

Files are copied and created via Aegis so that Aegis knows which files to transfer from your working directory into the baseline. If you don't tell Aegis, they will not be built, and later they will not reach the baseline.

3.3. Preview and Review

Once your Web pages build successfully, it is possible to see what the public will see - to preview them. We do this by running a specialized web server that "overlays" the files in your development directory over those in the baseline. Essentially, this is a simple search path. The specialized web server tell you a URL to open to preview your changes, and it will exist until you interrupt it, usually with Control-C.

Reviewers may use the same technique to look at an author's change, however reviewers are also encouraged to look at the HTML itself.

Using this specialized web server, it can be established exactly what the public will see when the pages finally arrive in the Web baseline. All of the Web pages are available in this way, as the change's pages are seamlessly integrated with the baseline pages. In this way, it is only necessary to copy the pages you actually need to edit. All other pages are obtained from the baseline. This often represents a large saving in working disk storage.

We have a set of layout guidelines which are available internally for our Web developers to read. Reviewers (theoretically) review against these guidelines, and developers are encouraged to have a quick look at the guidelines before

completing the development of their changes.

3.4. Integration

The integration stage is not particularly different from the integration stage for DNS. However, some of the more time consuming tasks, which are not of great interest to developers are done at this time: mostly generating or updating the various indexes for our Web as required for the specific change.

3.5. Concurrent Development

Performing file copies via Aegis allows Aegis to note the version of the file at the time it was added to the change. This version information is used later to detect file conflicts, and to merge file contents.

The chances of having Web pages in common are fairly low, however it happens fairly regularly. The use of separate development directories means that this can be ignore for as long as the author wants to. When it is time to complete development of a Web change, the versions of the files are checked by Aegis. If any are out-of-date, a 3-way merge may be performed to merge the Web baseline updates with the edits in the change.

3.6. Results

The AGSO Web is intended to be a "publication on demand" medium. As a result, any data the public can see needs to be of publication quality already. It is then only required for the client to print interesting material one the client's printer. While the HTML formatting may leave a little to be desired at present, we believe that the process we employ is more than capable of delivering the quality of content that we desire.

4. Summary

Many of the processes we perform on computers involve manipulating and assembling files in some way, many can be managed using SCM techniques. Even where significant manipulation (e.g. compiling software) is not required, automated input validation helps reduce the number of avoidable errors (much as compilers validate against input language).

The other advantage conferred by Aegis is that privileged operations are recorded, and are available for auditing purposes. The privilege is controlled by access control lists, and requires active conspiracy by more than one person to subvert. Even then, the audit record is inaccessible. By using Aegis' notification facilities, it is even possible for highly privileged system tables to be updated, and yet *root* passwords need not be disclosed.

The use of Aegis to assist in the management and availability of critical system services has been very successful, and will be used again.

The use of Aegis to ensure that our Web is of high quality has been equally successful, and is a model we would happily recommend.

5. References

- Miller, P. A., *"Aegis Is Only For Software, Isn't It?,"* AUUG '96 ACT Summer Conference Papers, 1996.

[1] Miller, P. A., *"Aegis - A Project Change Supervisor,"* AUUG '93 Conference Papers, 1993, p. 169-178.

[2] Miller, P. A., *Aegis User Guide* `http://aegis..sourceforge.net/`,

[3] Mockapetris, P., *"Domain Names - Concepts and Facilities",* STD 13, RFC 1034, USC/Information Sciences Institute, November 1987.

[4] Mockapetris, P., *"Domain Names - Implementation and Specification", STD 13, RFC 1035, USC/Information Sciences Institute, November 1987.*

Aegis is Only for Software, *Isn't It?*

Peter Miller
pmiller@opensource.org.au

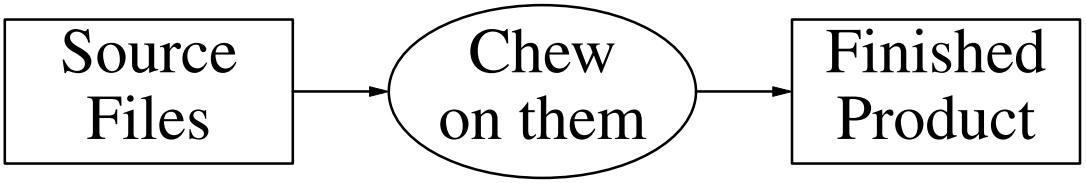
- Aegis at a Glance
- Managing DNS with Aegis
- Managing WWW with Aegis

Aegis at a Glance

- Software Configuration Management

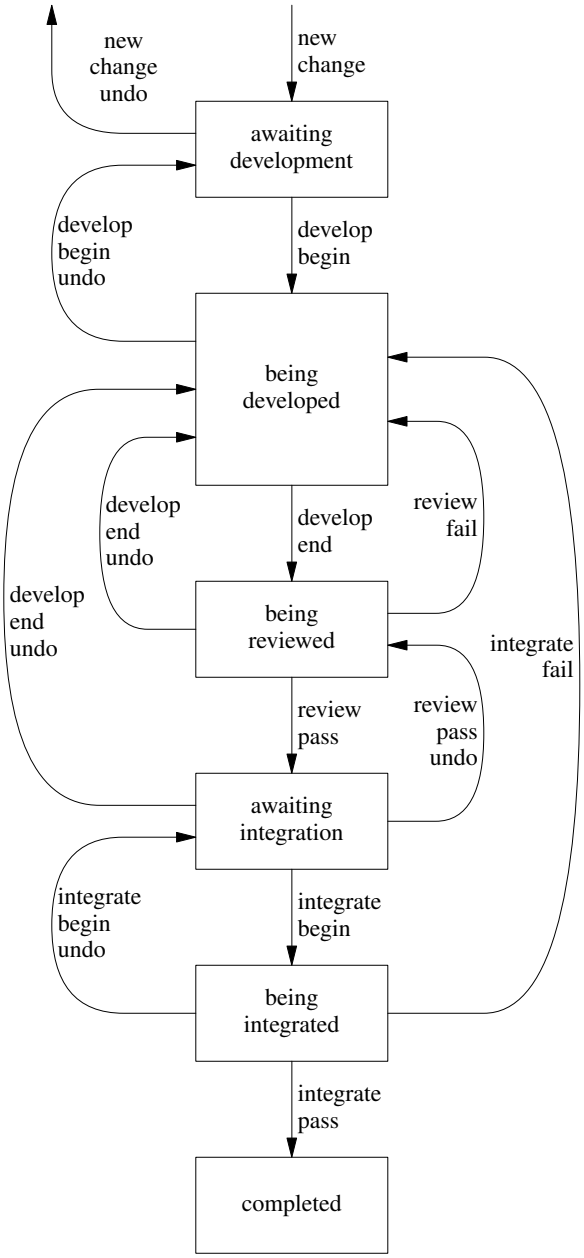
  Manifest control, Version control, Build control, Change control, Quality control.

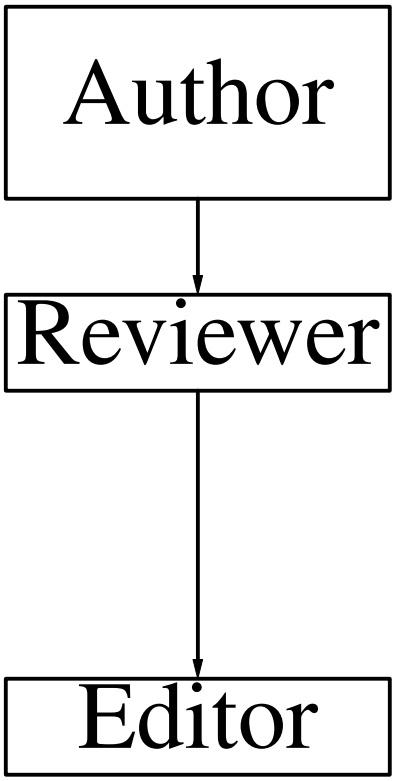
- Software development analogue



- Aegis provides change process and change environment

The Process in Too Much Detail





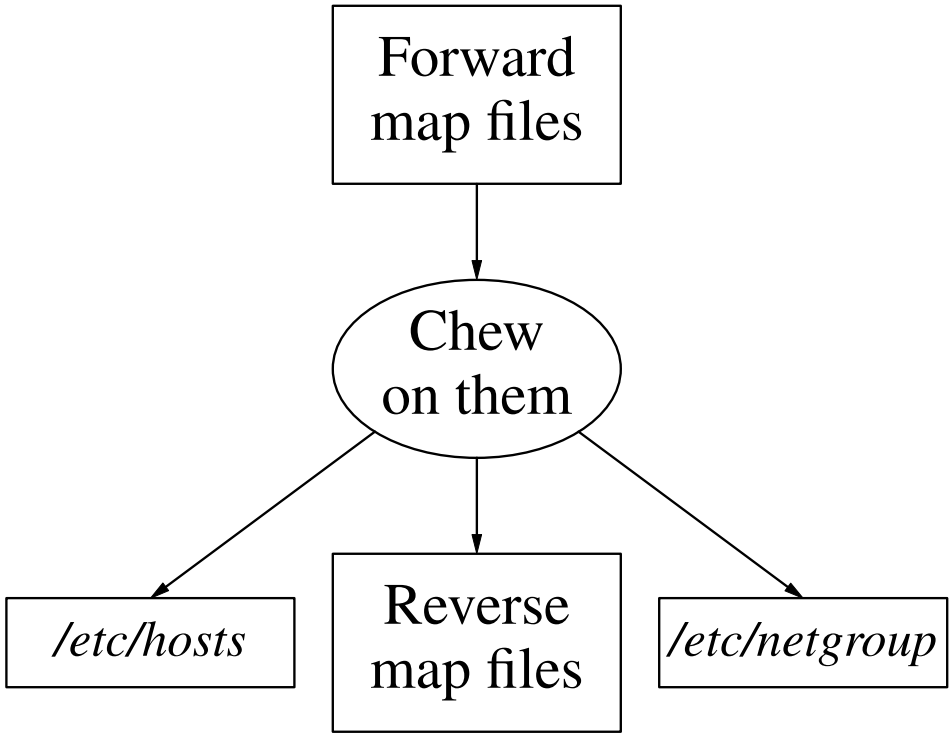
Advantages of the Process

- Access control lists
- Insulated development areas
- "Backing Out" is Easy
- All state transitions recorded
- State transition notification

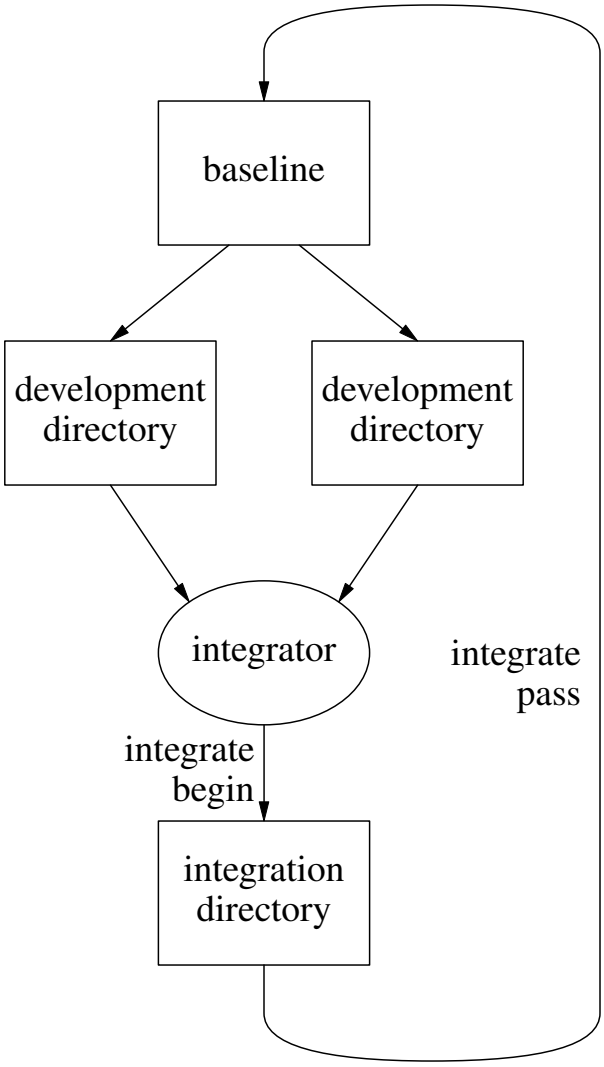
Managing DNS with Aegis

- Users logged in as themselves (ACL)
- No *root* access required
- Fully audited and reportable
- Notified of all changes
- Coordinates multiple system admininstrators

The DNS Build Step



The DNS Integration Step



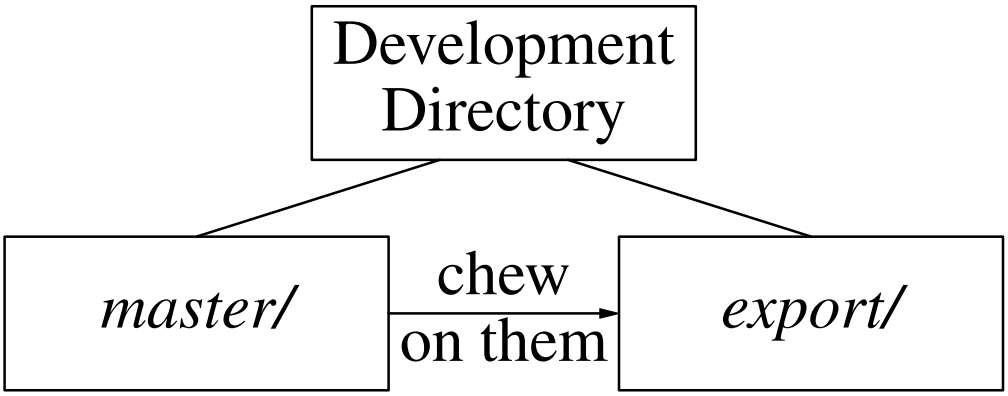
- Conflicts detected and managed
- This is when history is updated
- No *root* access required for *named*

notification

Managing WWW with Aegis

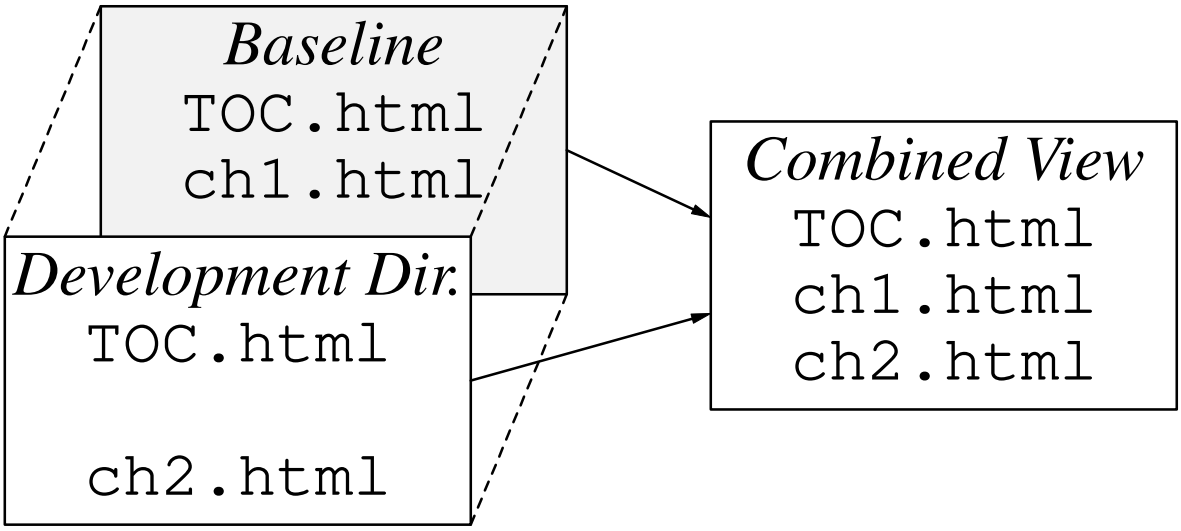
- Users logged in as themselves (ACL)
- Fully audited and reportable
- Notified of all changes
- Coordinates multiple authors
- Peer review is well understood

The WWW Build Step



- Server-side includes are resolved
- Include dependencies automatic
- HTML is checked against DTD
- Reports generated, e.g. *What's New*
- SQR compiled, *etc, etc*

Directory Stacking



- Only files being edited or created are in development directory.
- Build tool must be able to cope with directory stack.
- "Test Server" must be able to stack, too, so can preview.

The WWW Integration Step

- Integration build also constructs the indexes and other files not normally interesting to developers.
- Same picture as before, integration directory renamed to be the baseline.
- Conflicts are detected and managed.
- This is when history is updated.
- Server is sufficiently lightly loaded to run from *inetd*, so no notification required.

Aegis is Only for Software, *Not!*

- Access control lists, no *root* required
- Audit trail, statistics and reports
- Quality control for Publication on Demand
- Quality control for critical systems

// vim: set ts=8 sw=4 et :