Refactoring the Publishing Process

Steve Loughran
HP Laboratories
Filton Road
Bristol BS32 UK
slo@hpl.hp.com

Erik Hatcher
Darden Graduate School of
Business Administration
USA

October 27, 2006

Abstract

This paper dicusses how the technical book publishing process could address time to market and eBook issues through the adaptation of collaborative software engineering processes to the entire publishing process, from the initial authoring to the final preflight stages.

We identify where, based on our own observations, the current process is flawed, then explore how it can be improved. While some of our suggestions are merely the adaptation and customisation of standard software development practices —source code management (SCM), defect tracking, testing and continuous integration, we also explore how research in Computer Supported Cooperative Work (CSCW), in particular the rôle of annotations in reading, could be applied to the process to significantly enhance the value of eBooks.

1 Introduction

This paper analyses some of our experiences of writing a computing book [EH03], observing where the current process is not up to date with modern software development processes. It is our premise that the adoption of standard software project tools will improve the time to market of books, and equally importantly, reduce the effort required to produce revisions. For too long the book industry has adopted a waterfall process, of author, copy editor, typesetter, proof-reader to printer, a process which only works if the frequency of editions is low enough to bear the costs, the technologies being written about are stable, and the volume of sales per edition is high.

We believe that a process in which all participants on the project collaborate more tightly, using a unified file format that all can edit, would reduce costs, yet also enable new updated editions of a book to be released more easily. This would act as a foundation for new business models for publishing, potentially making on-demand printing and eBooks viable propositions.

2 Competitive pressures

In 2001, the total sales of computing books is significantly down from 2001. As book sales are lower, so the return on investment from a normal book is less, unless the costs of producing books is reduced. Yet the publishers still have to introduce new books every year as technical changes render the existing copies obsolete. It is rare that a timeless classic such as *Design Patterns* [GV+] can provide year-on-year revenue without updates being needed. More often even a successful book, such as *Java in a Nutshell* [nut97] needs updating whenever a new version of the product is released. If the lifespan of a book is the same, yet sales are down, then margins are less. The non-recurring expenses (NRE) must be spread across less books, pushing up their costs. Lower sales of books forces a higher actual selling price (ASP) just to cover margins. There is evidence of this in the recent price changes in O'Reilly Press books, adding $10 to a book originally priced at $24.95 [JB02]

2.1 The growth of the online bookstore

It would be interesting to discover what percentage of books are being sold through on-line versus the traditional retail channel. The discounts that on-line sellers can offer buyers can be significant to offset the costs of postage, saving buyers money over the local store. They also provide the benefit of coverage; anyone with a postal service can obtain books that may be too much of a niche for the local area. For The higher price and lower volumes of technical books make on-line purchasing of such tomes more attractive than for commodity literature, so one can hypothesise that the percentage of technical books sold on-line may be greater than the norm.

As the on-line retailers are still struggling to make a profit, they must want to increase their margins, which comes down to demanding a lower price for their books, more favourable payment terms, volume rebates or market development funds. In the publishing industry, the latter would translate to cash in exchange for better placements and promotions on the site. With the ongoing consolidation in the industry retailers will be putting continual pressure on the publishers to lower their prices.

A subtle effect of the on-line store is that it is starting to change how books are written. When selling a book in a store, cover art, title, and spine are all important to attract the attention of potential customers, but then they can browse through the book to see if it has value. Indeed, bookstores such as Borders welcome customers reading unpurchased books in their in-store cafés, so that they can make informed decisions about a purchase.

In the on-line world, the entire text is not available for glancing through, unless a free ebook is made available. But we believe that Amazon does not encourage such a tactic. Customers will search in vain for a pointer to the download site of Thinking in Java [Eck] on their page dedicated to selling it to potential readers [1]. Clearly some people in the on-line channel do view free eBooks as a competitive threat.

Instead of free eBooks, retailers want good cover art, a title, a table of contents and the back page material, The table of contents and one or two sample chapters have become the substiute for the ability to browse; user reviews the nominally impartial advice. A good book for online sales must have a table of contents that demonstrates to the purchaser that the book will contain the information they want, in the absence of the ability to flick through the book. The book must also have a title that not only conveys the value of the book, it must contain the relevant keywords to ensure that it appears during a search. Hence our inclusion of of Java alongside Ant in our book's title [EH03].

2.2 The failure of eBooks

The eBook market has not taken off. The most successful rôle for the books has been as a preview to the printed copy, such as with Mastering EJB. [$\mathrm{R}^{+}01$]. Placing non-printing previews up on busy developer web sites may cannibalise some sales, but the emergent conclusion is that the potential customers gained outweighs this. Publishers are not cannibalising sales if the people who read the eBook edition would not have seen the book in the first place.

The reason to why eBooks are unsuccessful is still not determined. One possibility is that the Digital Rights Management infrastructure put in place to prevent 'piracy' acts as a barrier to adoption. Would readers buy a book if they could not view it on all their computers, or move it with them when they upgrade their system? Some people [O'R02] believe that this is a possible cause. Another is that paper documents easier to read, navigate, annotate and hence understand and use. Sellen and O'Hara provide evidence supporting this hypothesis [SO97]. Finally, paper books provide a tangible sense of value in exchange for the money spent.

We would like to introduce an extra hypothesis, one proposed by Barry Brown while investigating mu-

[1]http://www.amazon.com/exec/obidos/ASIN/0130273635/

sic sharing [BGS01]. He suggested that the CD collection provides a visible display of taste and style to visitors, as well as providing tangible proof of ownership of the 'gold copy' of music. Similarly, the office bookshelf full of technical books does not merely act as a reference collection, it advertises the skills and interests of the reader. A developer with the Perl and Linux manuals is obvious a Linux and Perl expert. A developer with Java, Ant and EJB books is a serious Enterprise Java developer. And someone with Design Patterns [GV^{+}] and Refactoring [Fow00], is clearly a rigorous developer, regardless of the underlying technologies they choose to work with.

A collection of eBooks on a hard disk does not provide the same tangible advertisment of skills. Unless a means to subtly provide such a display is provided, such as an automated means to generate a 'books I have' web page; or a 'books I own' poster, printed books will retain values above and beyond the transfer of knowledge. Barry also observes that sharing is an important function of ownership of music: likewise, printed books are regularly borrowed and returned. The anti-piracy mechanisms intended to prevent widespread duplication of content, have the adverse effect of preventing lending, a form of sharing that can often lead to sales.

Another issue is simply cost: if eBooks are sold as an alternative to a paper book, then all costs other than printing and distribution still need to be covered: author royalties, editing, typesetting and proofing and marketing. Futhermore, if they are to be sold through the on-line booksellers, these organisations will demand an equal (or perhaps greater) discount to their paper books. Hence although the replication cost of an eBook is zero, the total production cost is quite high.

This is not to say that eBooks do not have value; we strongly believe that an on-line copy of a book should be something all purchasers of the paper book should somehow gain access to. Such copies can be kept on laptops or on different development systems for easy portability, and offer searching and cut-and-paste opportunities that the printed books lack.

2.3 The challenge of open source

The growth of open source introduces opportunities and challenges. The opportunity is that an open development process makes it easier to see the internals of a project; there is no need to get insider access to a project as is the case for closed source programs. In such closed-source programs, inside access to the company is needed to write books such as [Gun01]. This places competitors at a significant disadvantage, and can threaten the independence of the authors: if they don't write what the software house likes, they won't be able to write a followup edition.

In the open source world, anyone can write a book. This is good, although one still needs to write a good book. Also, the free documentation generated by the projects is actually one of the main competitors. This is why we chose to write a book to supplement the Ant documentation, rather than rewrite it.

One project, JBoss [FS02] derives much of its revenue from the commercial book. This has caused them to no longer maintain the on-line documentation: they could not afford to. This leads to an interesting situation —any book that competes with [FS02] actually derives the team of revenue, so unless the publishers somehow share royalties with the developers, there will be animosity.

Similarly, the Free Software Foundation have a long stated objection to commercial publishers writing books on their GPL licensed software unless released under equivalent terms[GNU99]. From their perspective, copyleft documentation is the only way that people can take full advantage of their right to change the code and redistribute those changes; developers need an equivalent right to change and redistribute the documentation.

The big challenge of working with open source is that after the book is published, it dates much faster. In the closed source world, version 2.0 of a product may have begun before version 1.0 is finished, but nobody outside the development team can see it. To the outside world, version 1.0 remains the sole version of a production until version 2.0 ships.

In the open source world, the fact that version 2.0 was underway is obvious to all users, and indeed whenever any issue with version 1.0 is raised in the support system, the usual response will be "have you tried tonight's build?". As new features are continually being added to the open source project, the longer a book is on the shelves, the more visibly out of date it becomes, even before the next release is formally available. Interestingly, some of the changes are a direct result of the book itself. To cite an example of this from our own book, we had to use a contrived trick to determine hostname of a local machine for deployment purposes, a trick that does not work on MacOS X systems. The next version of Ant will have a `<hostname>` task we have written to do this elegantly, providing users with an easier mechanism. So we must effectively obsolete our own book in the long term interest of Ant users.

There is no way to address this aging problem other than timely releases of books —as soon as a new version is issued, combined with continued tracking of changes. These changes need to be reincorporated into the overall package of the book. One tactic would be to say nothing, then release an updated version of the book when the next official software release is made. This could maximise revenue to the publisher, but minimise benefits to the reader. An alternative approach is to provide a "what has changed" document on the web site. This shows the users not what the errata in the book was, but what was no longer valid or necessary in the current development build. A monthly release of this update would require effort from the authors, but it would benefit users and perhaps help create a sense of community.

A third tactic would be one of continuous development and integration: every month the eBook would be upgraded to track changes. Registered owners of the book would receive email notification, so could download the latest version. The nice thing about this tactic is that it also ties in with on-demand-printing: new print runs would be with the latest version, and it would even enable a subscription model for eBooks, providing a tangible value over static printed content. Unfortunately, issuing a new, updated eBook every month or quarter is simply not possible with today's book production workflow. It could fit in with the workflow of books as they are written, helping create advance publicity and generate feedback from all the readers, who are now acting as reviewers.

Regardless of the actual update process, if the only way to work with open source projects is to frequently release updated versions, the secret to do so profitably must be to keep non-recurring expenses low, and time to market short.

3 Collaborating via eBooks

We have already discussed how eBooks do not currently offer enough tangible value to be popular, and observed that one area that they could improve is in being more up to date compared to printed books, provided the author(s) continually updated them.

How else can eBooks be used to give them an advantage over paper copies? The answer is that they can be used as a centre of a collaboration tool. The book reader software must be able to support annotations, a feature that is not available on the free Acrobat PDF reader, but only on the full-blown product. This is despite (or because of) the fact that Sellen and O'Hara demonstrated that free form annotation is an an essential part of technical reading [SO97].

For eBooks to be as usuable as paper books, not only must the navigation be improved, the standard reader tools must support annotations. If the user has copies of the book on their different systems, they also need to be able to keep the annotations synchronized across the boxes.

A profound opportunity arises if annotations can be shared across owners of a book, either in a local site, or over the Internet as a whole. If one user clarified a section in a book, or disagreed with it, that clarification or comment could be made visible to other readers.

This use of an eBook as a focus of CSCW has been explored in the past, with one of the key barriers the robustness of annotations [B$^+$01]. If all annotations are lost when a book is updated, there is little value in using the annotation system.

We don't provide a solution to this problem, but observe that a production quality book can be written to a more rigorous process than normal articles, and this process could involve the assignment of a unique ID, such as a UUID, to every section, paragraph and sentence of a book. Annotations could be bound to these objects, and so remain with the relevant text even as that text moves around. We would go one step further and suggest that if every such entity in a book was given a URL, then RDF metadata could be used as the format of annotations, and it would be easy to email pointers to books around. The URL of each element would also represent the home location of each element on the web, the place for all shared annotations to exist.

4 A new workflow

A key observation from our experience was that many of the people involved worked from home. For the authors, this was because we were writing this book outside our working hours. Yet the other participants in the workflow also worked from home, because it suits everyone With all inputs and outputs of the process text and data files, there is no need for physical locality, and there is no compelling need to work office hours. If the copy editors and proof readers work from home, they get a lifestyle they can enjoy, and the publisher does not need the overheads of providing office space. It also lends itself to an industry where each person is self-employed, collaborating with others for a single job, then moving on to different projects, perhaps with different publishers, as their stage in the process is over. Again, this can help the publisher, as it keeps fixed salary costs down.

As it stands, the people involved in the publishing process are an ideal target for CSCW technologies. The problem of generating quality books is highly collaborative, and yet there is no real framework for the remote participants to co-operate. In our own case, email and FTP were the primary channels for communicating and sharing with the rest of the team, primitive technologies that come with failings that modern systems do not have.

No version control. Without the versioning aspect of SCM, and with documents being sent by email and ftp, it is easy to get confused as to what version is live. The publishing team all used naming rules to avoid this, but still we encountered occasional version conflicts as a result of accidental use of old copy.

Email is the wrong channel for binaries. Large binary files should not be sent as email, as it is too slow over dial up links, links that some people in the team used at times. Better to share the files via a server and send links to them.

No group history. Email without a list server lacks a repository of discourse. We needed a shared, searchable email repository, so that people joining the group could catch up on past work.

No defect tracking. The process needs a more rigorous method of submitting corrections than email notes or attachments to PDF files. Errors in a book are as critical as bugs in software. They should have defect tracking and maybe automatable tests.

Our proposed solution here is to take the standard software development tools and many of the processes used for open source software development and apply them to the publishing process, treating it as a variant of a classic software project.

1. iterative development
2. automated build process
3. SCM for version control.
4. mailing lists as the email channel
5. defect tracking with a defect tracking system
6. a project page within a larger portal site

It would also be possible to use a closed source workflow system such as Lotus Notes or Microsoft Exchange, but we do not believe that these work well in the highly distributed and transient workforce of a book publishing team.

The advantage of using the tools of the software development teams, is that they can integrate with the process used to build and run the software accompanying with the book. More importantly, one can use an automated build tool to create PDF copy of the document on a nightly or hourly basis.

4.1 Iterative development

The current workflow is nominally a waterfall: authors hand-off their work to the editors, the editors hand off the copy to the typesetters, the proof-reader proofs the typeset copy and then it is published. The authors still need to review the copy after every stage, and to stay synchronised with the Ant 1.5 release, we were making changes to the copy right up to the day that Ant1.5 and our book were both officially complete.

The only way such constant change can be managed is to move to a process where everyone is working with the same file format, and where everyone can —if needed— make changes to documents at any stages of the process. This does not mean that they must make changes. Just as with any software project, the closer the product is to release, the more careful consideration each change required. Nor are any changes made without informing the others; this is a required feature of the SCM system.

4.2 Configuration management

A shared SCM repository is essential to this process. We would argue against CVS, as its limitations are well known, especially for non-text files, and it is wrong to change your text editing format just because the SCM tool is limited.

What we think is the most appealing is the forthcoming Subversion tool [?], because it is designed to retain the best features of CVS: support off-line working and slow network connections, and offer a WebDAV interface. The WebDAV extensions to HTML work through firewalls, and provide secured write access to files; if https: connections are use the access is also encrypted against listeners. Many tools are starting to support WebDAV as a remote access mechanism, including FrameMaker. These tools may be able to connect directly to a Subversion server.

One barrier here is the need to educate everyone to use SCM rigorously. We assume that software authors do not need to be so educated, but other techical authors may need to be introduced to the benefits; the same for the rest of the team, who will need to learn how to use it properly. An easy to use tool (again, not CVS), is important.

A subtle side-effect of mandating the use of a publisher-supplied SCM server from the outset of the project is that the publisher will be able to track progress of the project between milestones. Although they may choose not to examine the work between the milestones, a lack of changes to the SCM database would be a sign that an author was not working on the book, and that a discreet "how are things going" email was in order.

It also provides a back up of all the documentation, reducing the risk of loss of work. We went through a notebook and a hard disk during our year's work, but lost nothing as we used our private CVS server.

4.3 A project start page

A project start page would act as a central focus for the project. It would list the status of different people, the current schedule and provide access to the defect tracking and repository information. It could even link to the online status of people's instant messenger accounts.

The purpose of the page would be not only to benefit the team, but to provide status for those people depending upon the team, specifically the rest of the publishing staff, who could now see the status of the project without bothering anybody.

4.4 Annotating

If reviewing and editing is taking place in parallel, how can one reconcile review comments with text? Page and line numbers are inadequate in a continuous integration model, unless you use the appropriate snapshot of the document to determine the revelevant text in the current version. Version-independent annotation would seem as critical in proofing and review as it would be for end users.

Reviewing adds one more requirement; for a bug list to be visible, showing the status of comments; whether they had been addressed or not, and what the resolution was. Such a defect tracking system is considered foundational to a software project; the application to defects in a book is merely an extension. We do of course, benefit from version-independent references to the annotations; a URI or other identifier of the book element under discussion. A customised book defect tracking system could assist by displaying the contents of the identified element from the version of which the defect was reported, alongside that of the current version.

4.5 Testing

It may seem odd to have testable defects in a book, but is not really. If the defect report includes a phrase that should not be in the book, then an test can be automatically be generated from this phrase.

What one can not test in a book is much in the way of functional tests: that the book is actually correct. It may be possible to extract the code listings from the book and compile or execute them, to verify that they are error free. It may also be able to write tests that a chapter must contain keywords, such as the names of concepts, or in our subject area, Ant tasks.

A more advanced concept would be to use Natural Language Processing to validate sentences. The grammar and style checkers of MS Word do this.

4.6 Reader involvement

Should readers be involved in this process —and if so, how? Manning have set up a work in progress site, thecodercoop.com to show previews of chapters of forthcoming books. The belief must be that showing forthcoming work reduces the risk that readers will by competitive books.

The broad range of skills of readers would actually make them great technical reviewers, a point noted by Eckel [Eck]. This is also one of the likely reasons behind the on-line development process of the NetBeans guide: [Net02].

One problem of opening up your book to many reviewers is that the review process gets that much more complex, *unless* you have a good defect tracking and annotation system in place. Without these the handling of review comments is significantly more labour intensive.

We feel strongly that reader involvement, especially in books related to open source projects, is the way forward. We merely need the infrastructure to make such involvement manageable.

4.7 Merging changes

How can the team merge changes made simultaneously to a document? Whenever this happened to us using MS word, it was viewed as a near-disaster that we had to struggle to recover from . In open source projects, parallel changes are viewed as an inevitable occurence, so the SCM tools are designed to resolve most changes as seamlessly as possible.

Using plain text files, or another format where reconciliatin is tractable is a primary first step. Even there, careful proofing of merged documents is essential, so steps must be taken to minimise the scope of such conflicts. A solution from very large C++ projects would seem to apply here: move to fine grained storage of content —in the C++ world, this means implementing only one function or method per source file. In the book domain, this means that instead of having a file for every chapter, authors use a file per section, or per subsection. A granularity of the individual paragraph is actually conceivable, especially if the elements are stored in some kind of database, rather than as text files.

If each subsection were a standalone element, then a chapter would be described as a tree of sections and subsections; a change to the ordering of these elements would occur in the chapter document, but not affect the change history of the contained elements. Likewise, a change in a subsection would not propagate up to the containing elements.

The biggest barrier to this model is how to seamlessly integrate it with text editors. The chapter per file model works because it is a good size for much editing; finer grained entities would be hard to edit unless the editor could display the collection of entitities as a single coherent chapter or book.

4.8 Automated build process

If the PDF copy can be made automatically from the source documents in SCM, then a system such as CruiseControl [MF00] can build nightly or hourly versions of the PDF. These could be combined with JDF job ticket [cip01] and sent via SOAP calls directly to an offset printing process for on-demand publishing, or deployed to a web site for public viewing.

If the defect tracking system automatically generates tests, the automated build can also run all the tests against the copy, verifying that all defects marked as FIXED are fixed, and indicating which open defects can no longer be found in the source. This could be displayed on a status page on the web site.

The initial benefit of this process is that because it is so easy to create PDF files for review, more regular review copies will be generated. If an eBook publishing model is used in which regular updates of the eBook is a core value proposition, then the automated build will be the foundation for these regular updates. The value of an automated, hands-free build process then becomes one of keeping costs and errors low.

4.9 Managing References

It is interesting to note that the references section of a book is still primarily in a pre-web era, possibly with good reason. We did have a table of web URLs, but of course, in the twelve months it took to write the book, approximately a quarter of the URLs were no longer valid, and new links had to be found or the references deleted.

The printed publications list had a style dictated by the Chicago Manual of Style [Chi], a manual that is yet to adopt URLs or recognise that ISBN numbers are more useful as search terms than the city of the publisher. We need a way to unify paper and web publishing in a way that doesn't suffer from broken links

The solution is nominally DOI [DOI00], which uses the CNRI Handle System to resolve DOI handles to URLs of either on-line content or an information page about printed content, could be the solution. The terms of DOI handles (an annual $1000 fee and a per-handle charge) bias the system towards traditional publishers, rather than individuals publishing their own pages. To index open source documents, we would need a low cost yet compatible alternative, which could be accomplished by setting up our own handle system resolver, under the auspices of Apache or another major site in the open source community. These would not be DOI handles, but they would be resolvable in the same manner.

The alternative is use Google search terms as a resource location service, which is effectively what users have to resort to when the original URLs decay, and/or to list these resources on the web site accompanying the book, instead of on paper. Of course, that becomes another maintenance effort, unless the web page uses google search terms for its resolution. An automated process using Google's new SOAP API could perhaps keep a resource page up to date, or notify the maintainer when resources had moved.

5 A vision of the future

Our vision of the future then, is the following.

A publisher runs a portal web site, which has two aspects: the view for the publishing team, and the view for readers.

For the writers, the portal is the access point for the collaboration with the others in the team. The SCM-managed document is accessible here, the archive of all mailing list discussions is stored for searching and retrieval. Everyone works using the same toolset -perhaps the publisher has a floating set of thirty Adobe FrameMaker licenses which can be granted to any authenticated user across the internet. This editor is used not only for changing the document, it is used to select and annotate defects. Macro code added to the editor provide a defect logging facility that automatically generate defect reports *from inside the editor*; these defects are stored on a server database along with location information. As semi-structured reports are used, the reports can be used to auto-generate tests that determine whether the erroneous text is still present in the document, or what text to add is still absent.

On a regular basis, perhaps twice a day, a new set of PDF files are auto-generated from the document source, and the test code then determines which logged defects are still present in these files. The results of these tests are made visible to all in the team via web status pages. There is an implicit benefit to this to the publishing house: they will be able to track progress on a book without having to approach the authors.

The PDF files are then made available to the reader side of the portal. Here registered users are given early access to the documents as they are written, in exchange for providing feedback. To keep feedback manageable, chapters-in-progress can be locked out; readers will be notified by email when the chapters are ready.

When the book is published, the current snapshot of the book is sent to the publisher. The runs may be smaller than today, so perhaps a VDP process is used in which the print job is automatically submitted to a JDF print process [**?**]. As the books are printed and bound, priority volumes are sent to key publishers, such as Amazon and the influential specialist bookstores -SoftPro, Powells and the like, while the main volume of books can follow.

At this point the eBook of this edition is also made available in a single download. Purchasers of the printed copy automatically gain access to this through some registration process; if the eBook version contains any DRM restrictions then there should be no limit on how many copies a purchaser can download, including copies to give away to friends and colleagues.

Readers of the eBook can not only add their own comments to the book, they can publish the comments to a nominated 'group'; these comments are stored back on the server and made visible to all other members of the same group. This permits private groups to have their own private set of annotations, perhaps stored on their own private annotation server.

During the lifespan of the book, Authors will be expected to maintain it. Their maintenance will be without major input from the Copy Editors and Typesetters, who will primarily only get involved when a new edition is released. Interim releases; repeat runs of the first 'edition' can be derived from the previous published edition with all the changes the authors have applied. Clearly this implies that the authors must have have adequate grasp of the written language to produce readable content, and the typesetting system to be robust against extra or updated content.

Is this a realistic vision? Perhaps. Even without moving towards a a model in which a book is updated and released more regularly than today, moving to a collaborative authoring and publishing process should yield immediate and tangible benefits.

6 Conclusions

We have examined some of the issues of authoring and publishing a technical book in the context of today's business climate, in which online booksellers, the challenge of writing for today's short-life software products, especially open source, and the unrealised potential of eBooks. Our premise is that a CSCW collaboration environment supporting some of the foundational processes of modern software development to would allow the publishing industry to deliver books more easily than at present.

Once such tools and processes have been adopted, new delivery mechanisms and lifecycles could be explored. In particular, moving a more regular publishing of eBook editions, coupled with a collaborative annotation system, could add enough value to eBooks that the format may become viable.

7 How we built this report

This report was generated using LaTeX, creating the pdf version directly from Ant;

```
<target name="pdf" depends="init">
  <exec executable="pdflatex"
      failonerror="true">
    <arg value="-silent"/>
    <arg value="-c-style-errors"/>
    <arg value="-interaction" />
    <arg value="nonstopmode"/>
    <arg file="refactoring_publishing.tex" />
  </exec>
</target>
```

It would be trivial to include in a continuous integration process, in which an automated tool such as CruiseControl [MF00].

References

[B$^+$01] Brush et al. Robust annotation positioning in digital documents. In *Proceedings ACM CHI2001*, 2001. doi:10.1145/365024.365117. 4

[BGS01] B. Brown, E. Geelhoed, and A. Sellen. Music sharing as a computer supported collaborative application. Technical Report HPL-2001-103, HP Laboratories, 2001. http://www.hpl.hp.com/techreports/2001/HPL-2001-103.html. 3

[Chi] *Chicago Manual of Style*. 8

[cip01] *Job Definition Format*, 2001. http://www.cip4.org/. 8

[DOI00] DOI. Doi handbook, 2000. doi:10.1000/186 http://doi.org. 8

[Eck] B. Eckel. *Thinking in Java*. http://www.mindview.net/Books. 2, 7

[EH03] Steve Loughran Erik Hatcher. *Java Development With Ant*. Manning press, 2003. http://manning.com/antbook. 1, 2

[Fow00] Martin Fowler. *Refactoring*. 2000. 3

[FS02] M. Fleury and S. Stark. *JBoss Administration and Development*. SAMS, 2002. 3

[GNU99] GNU. Free software and free manuals, 1999. http://www.gnu.org/philosophy/free-doc.html. 3

[Gun01] E. Gunnerson. *A Programmer's Introduction to C#*. Apress, 2001. 3

[GV$^+$] Gamma, Vlissedes, et al. *Design Patterns*. 1, 3

[JB02] Tilley J. and E. Burke. *Ant: the definitive guide*. O'Reilly Associates, 2002. 2

[MF00] Matthew Foemmel Martin Fowler. Continuous integration, 2000. http://www.martinfowler.com/articles/continuousIntegration.html. 7, 9

[Net02] *NetBeans: The definitive guide*. 2002. http://www.netbeans.org/about/books/. 7

[nut97] *Java in a Nutshell*. O'Reilly Associates, 1997. 1

[O'R02] T. O'Reilly. Repeated misconceptions about eBooks, 2002. http://www.oreillynet.com/pub/wlg/1664. 2

[R$^+$01] E. Romans et al. *Mastering EJB, Second edition*. Addison Weseley, 2001. http://theserverside.com. 2

[SO97] A. Sellen and K. O'Hara. A comparison of reading paper and on-line documents. In *Proceedings ACM CHI1997*, 1997. doi:10.1145/258549.258787. 2, 4