第 17 章

NPCA の部誌執筆を支える技術

72 回生 Hinata

17.1　はじめに

皆さんこんにちは。72 回生の Hinata と申します。ちょっと前に中学に入学したと思ったら、いつの間にか最高学年となり、ついでに受験生になってしまいました。時の流れはとても速いですね。正直受験したくないです。はい。

さて、今回の記事のタイトルは「NPCA の部誌執筆を支える技術」です。「〜を支える技術」って書くと、なんか技術評論社の有名な技術書シリーズみたいな雰囲気が漂って、なんかカッコよくないですか?

私は 2 年前の文化祭のとき、4 月に入ってからプロジェクトマネージャ (PM) を先輩から (交代する形で) 引き継ぎ、去年も部誌 PM を務めました。今年は PM を後輩に譲り、校閲を担当しました。

この記事では、部誌に関してはそこそこ経験がある私が、部誌プロジェクトの技術的な話からプロジェクト管理の話までいろいろ説明しようと思います。

17.2　NPCA の部誌について

NPCA では 2012 年度文化祭から現在に至るまで、校内の部活の中でも先駆けて、部誌を印刷物ではなく PDF としてウェブサイト上で公開しています。今年でもう 8 年目です。

こういう取り組みをしているのには様々な理由があります。

- 分量が多いので、印刷する文化委員会印刷課の人にも (灘校では校内で部誌などを印刷しています)、製本する部員にも、来場者の方々にも、そして地球にとっても優しくない。
- 記事の性質上ソースコードや写真が大量に含まれるので、校内印刷だとどうしても読みにくくなる。データとして公開すれば、リンクとかを遠慮なく貼れるし、リンク先に飛びやすくなる。
- 文化祭に来た人だけでなく、来れなかった人にも見てもらえる。(ありがたいことに、この部誌を読んでいただいているエンジニアの方もいらっしゃいます)
- シェアが簡単
- なんかパソコン部っぽくてハイテクで素敵
- ~~締切が遅くなる~~

文化祭当日は、QR コードと URL を書いたビラをばらまいてアクセスしてもらうという運用でした。ビラは普通に校内で印刷しても問題ないクオリティのものができます。

また、当時は LaTeX を直接書いて執筆していました。1 人ならいざ知らず、共同執筆となるとやっぱり Word みたいなソフトはかなり厳しいので、まあ妥当な選択でしょう。

さて、部誌プロジェクトが転機を迎えたのは 2017 年度文化祭です。この年から、様々な取り組みが始まりました。

小冊子

来場者にとっては、ビラをもらってもアクセスするのにひと手間かかり、ちょっと障壁があります。紙で配ると目次だけでも目を通す人が多いことを考えると、ビラだけ配る方式だとどうしても見る人が減ってしまいます。

しかし、オンラインで部誌を公開するメリットはとても大きく、とても捨てられるものではありません。そこで、部誌を抜粋してサンプルとして配布することにしました。これなら、ちょっと読んで興味を持ってくれた人をウェブサイト上の部誌に誘導することができます。

まあ、その代償として製本作業が復活してしまったんですけど。しんどいね。

Re:VIEW への転換

2017 年から、部誌執筆に Re:VIEW というツールを利用しています。ちなみに読み方は「リ・ビュー」らしいです。ずっと「レビュー」って読んでました。

これによって、ウェブサイトで記事を直接 HTML として公開できるようになりました。他にもいろいろ恩恵はあったのですが、それは後述します。

17.3 NPCA 部誌執筆環境の紹介 (技術的な話)

バージョン管理とリポジトリ

NPCA では、Re:VIEW を導入する以前から GitLab(gitlab.com) のプライベートリポジトリを利用しています。最近は GitHub でもプライベートリポジトリを無制限に作れますが、当時は有料プランでしか作れませんでした。

部誌だけでなく、文化祭用の展示なども GitLab で管理しています。

執筆ツール

先述したように、Re:VIEW を利用しています。PDF 版は review-pdfmaker を使って LaTeX 経由で生成し、HTML は review-webmaker で直接生成しています。

LaTeX 直書きをやめて Re:VIEW を使うようになった理由はいろいろあります。

- HTML 版を簡単に出せる。
- マークアップ形式なので、LaTeX よりも書きやすい。
- もし機能が足りなくても、最悪拡張できる (`review-ext.rb`)

実際に使ってみた感想ですが、Re:VIEW には全く文句がありません。Re:VIEW よりもその背後の LaTeX がつらいです。でもこれは LaTeX を直接書いてても逃れられないので、どうしようもないですね。

もともと、Re:VIEW は技術系同人誌の界隈で広まってきたツールです。とある現役部員 (当時) が OB たちと同人誌を書いたときに Re:VIEW を使い、それが使いやすかったということで推薦され導入が決定した次第です。

確かに見方を変えれば、サークル (部活) で集まって本を書くわけですから、部誌は立派な同人誌です。実際、灘校のアニメ研究会にはコミケで部誌を頒布した実績がありますし、大学の技術系サークルで技術系の同人誌を出しているところも結構あります。複数の作者が各々の好きなものを書くアンソロジー的な同人誌を「合同誌」と呼びます。つまり部誌は合同誌です。

CI(継続的インテグレーション)

Re:VIEW プロジェクトをビルドするには、LaTeX 環境が必要です。今は TeXLive があり、各プラットフォームに対応するインストーラも存在するので、インストールはそこまで大変ではありません。

そうはいっても、いろんな落とし穴にハマる人は部員の中でも多いです。特にフォント絡みが厄介という印象があります。部員に TeX マスターがいれば話が違うのでしょうが、あれを執筆者全員にインストールさせ、ちゃんとビルドできる環境にするのはなかなか困難です。

そこで、NPCA では CI(継続的インテグレーション) を活用しています。GitLab だとプライベートリポジトリでも純正の GitLab CI が使えるのです。これを利用しない手はありません。GitLab CI というのは GitLab のサーバ側でビルドするタスクを走らせてくれるスグレモノです。何が「継続的」で何が「インテグレーション」なのかは分かりませんが、とりあえずコミットごとに PDF が生成されます。

さて、GitLab CI を利用するには Docker コンテナを用意する必要があります。

当初は vvakame 氏の docker-review を使っていたのですが、途中から使い始めたフォント用パッケージ (beramono) が足りなかったので、そのパッケージ (正確には collection-fontsextra) を追加した自作コンテナ (fork) に切り替えました。

CI の設定ファイルはこんな感じです。ちなみに、リポジトリのルートディレクトリには Re:VIEW プロジェクトがサンプルと部誌本編の 2 つあり、それぞれ独立しています。それに合わせて、ビルドタスクもサンプルと部誌本編に分かれています。

List 17.1: .gitlab-ci.yml

```
image: fabon/review:2.5

build_sample:
  script:
    - './makepdf sample'
  artifacts:
    paths:
      - sample/book.pdf
  tags:
    - docker

build_bushi:
  script:
    - './makepdf bushi'
  artifacts:
    paths:
      - bushi/book.pdf
  tags:
    - docker
```

List 17.2: makepdf

```
# !/usr/bin/env bash

set -eu
if [[ $#  -eq 0 ]]; then
    #  引数不足
    exit 1
fi

set -x
cd $1

bundle exec review-preproc -r --tabwidth=2 *.re
bundle exec review-pdfmaker config.yml
```

手元で PDF をビルドできない人も CI の結果を確認できるのでとても便利です。

しかし、部誌に画像がたくさん含まれているせいで PDF が重くなり、おまけにページ数も相当多くなるので、GitLab(Web) 上で確認できないのはちょっと大変です。どうにかならないものでしょうか。

Slack 連携

GitLab のコミット通知、CI の完了通知、Merge Request の通知などが Slack に流れるようにしています。これはめちゃくちゃ便利なのでとても重宝しています。

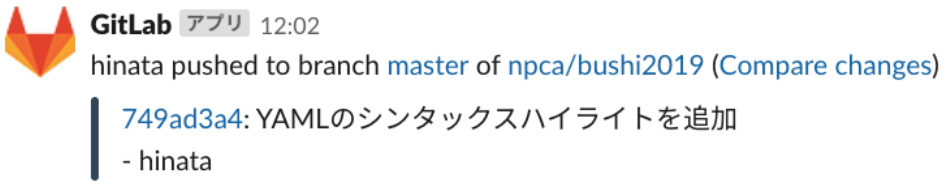

図 17.1: コミットを push したときの通知

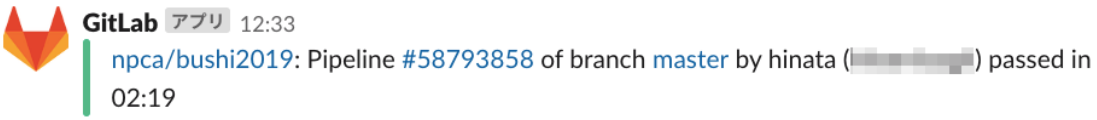

図 17.2: CI でビルドが完了したときの通知

今後の展望

技術的な環境整備については現状でかなり満足しています。

Re:VIEW で書いて GitLab で管理し、コミットごとに CI を回して PDF を生成する。CI の結果やコミットは Slack で通知されるし、完成した部誌はウェブサイト上で PDF や HTML として公開される。なかなかハイテクで素晴らしいじゃありませんか。

しいて言うなら、日本語の質を担保する何らかのツールがあるとなお良いかもしれません。文章チェッカーの textlint を導入するとか。

17.4　日本語を書く段階とリポジトリの運用について

執筆

執筆する時点ではブランチを切ったり Merge Request したりする意味は薄いので、全員が master に commit するようにしています。Git の素養を全部員が持っているわけじゃないですからね。執筆者は自分の記事ファイルしか編集しないので、 `git push -f` する大馬鹿者さえいなければ破綻することはありません。push できなかったら pull すればいいんです。

ちなみにわざわざ言及しているので薄々察している人もいるかもしれませんが、大馬鹿者は実際にいました。しかもよりによって、部誌執筆に必要な Git の基礎を教えるハンズオンで、上級生が「 `git push -f` するといいよ」と教えてしまったらしい。講師の選定はちゃんと考えましょう。

ちなみに「 `-f` すればええよ」と冗談でそそのかした別の上級生が元凶でした。冗談が悪質。しかしそれを冗談だと分からない人が Git 教えるのはちょっと……

あと、こうすると他の人の進捗が見えて、仕事が遅い NPCA 部員も多少焦ってくれるんじゃないか、という淡い期待もあります。~~実際のところ大して効果はありませんが~~

校閲

ちゃんと読める文章がまとまって、全体に関する執筆・編集が一段落すると、校閲の出番です。ここからは全体の構造を変えるような規模の編集はしません。文章の誤りだけでなく、Re:VIEW フォーマットの文法が正しくない文章を訂正したりもします。文章を細かく調整するので、校閲者が Merge Request を作り、著者に確認を取りながら修正していきます。全体がちゃんと読めるようになり、誤字脱字がなくなったら merge します。

ただし、これはあくまで理想的な話です。記事によっては完成するのが締切の直前で、そこから校閲していると間に合いません。もちろん部員が早く書いてくれればとても嬉しいのですが、これが現実というものです。それゆえに、記事が完成する前に、現時点で完成したところまで校閲することになります。別にこれでも問題はありません。ただし著者が Git を使えなかった場合、校閲した部分が全部吹き飛んで発狂することがあります。実際ありました。絶対に許さない。

17.5　プロジェクト管理と進捗と締切と過労死

個々の執筆者を除くと、部誌関連で大変な仕事が 2 つあります。校閲と PM です。

校閲者の仕事は、文章の誤りを正すことです。1 文字ずつ読んで誤字脱字がないかを調べたり、文章全体を読んで矛盾点がないかを調べたりします。余裕があれば内容の査読もやった方がいいと思います。記事の分量はかなり多いので、結構大変な仕事です。

今年に関しては、文章のクセも個性ということで、そこまで深入りした校閲はしていません。内容の誤りも、一応気付いた分は指摘しましたが、本腰を入れてやったわけではありません。

PM の仕事は、主に進捗管理です。つまり、全然仕事をしない部員たちの尻を叩くのが仕事です。出版社における編集者みたいですね。部員たちの仕事の遅さによって精神が削られます。他にも、たとえば LaTeX や Re:VIEW などをいじる仕事などもありますが、これはプロジェクト初期に一度やっておけば大丈夫です。ただし LaTeX は本当に難しいので、この作業を後に回すと精神が死んで LaTeX の愚痴をひたすら吐き出す機械になってしまいます。

2017 年と 2018 年は、私が PM と校閲をやっていました。上を見れば分かるとおり、どっちもそこそこの重労働です。おまけに、私はウェブサイトの更新も担当していました。こちらも部員とやりとりしてデータやスクリーンショットなどを集めないといけない、とにかく対人コストのかかるそこそこの重労働です。しかも文化祭直前は文化祭ページを更新しないといけないので、一

番忙しくなります。これを 1 人でこなすというのは明らかに無理がありました。

今年は校閲と PM を分割し、人が死なないようになりました。本当は仕事を分担するという発想がもともとなくて、単に校閲まで PM の手が回ってなさそうだったので引き受けただけなのですが、やっていくにつれて PM と校閲を分けることの素晴らしさに気付いたというわけです。やっぱり仕事は分担するべきです。

あと、締切をちゃんと守るという意識が薄かったのが問題でした。公開がウェブサイトなので、極論を言うと文化祭当日の朝に書き終わっても載せられるんですよね。そういう甘えは本当に良くない。

進捗管理どうする問題

NPCA ではかつて Redmine を利用していました。今も部内 Redmine は動いているものの、ほとんど使われていません。

現状だと進捗確認も Slack でやってしまうのですが、Slack はチャットなので進捗管理のために作られておらず、さすがに無理があります。Slack はどんどん流れていってしまうので、確認した情報を別の場所にまとめる必要がありますよね。

やはり Redmine を名実ともに復活させるべきなのでしょうか。ここらへんは部誌に限らず部全体の課題です。

17.6 おわりに

というわけで、NPCA の部誌執筆を支える技術の話でした。

もともとは技術の話だけする予定だったんですけど、プロジェクト管理に言及したあたりからチラシの裏みたいになってしまいました。正直なところ、今は「これ部誌でやる必要あったのかな……?」と思っています。

まあ、せっかく書いたので供養しておきます。同じような立場の人が読んで反面教師にしてくれるかもしれないので。

絶対にプロジェクトを炎上させないようにしましょう。約束ですよ。