

21時からの勉強会

その1



21時からの勉強会

21 時からの勉強会 その 1

21 時からの勉強会 著

2025-11-16 版 21 時からの勉強会 発行

前書き

隔週で有志で 21 時に集まって、わいわいと勉強したり、IT に関する内容について雑談しているサークルです。この度「技術書典に参加しよう！」という話となつたため、同人誌を出版することになりました。メンバ全員、同人誌どころか、書籍を作ることが初めてですが、今まで各位が培った様々な知識・経験を書いていますので、温かい目でご一読いただければ幸いです！

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

目次

前書き	2
免責事項	2
第1章 Raspberry Pi de 天体観測	6
1.1 まえがき	6
1.1.1 天体用デジタルカメラ	6
1.1.2 「天体用デジタルカメラ」とは一体どのようなものか?	6
1.1.3 どうして赤外線カットフィルターを改造すると天体用になるの?	7
1.1.4 スマート望遠鏡	9
1.1.5 「スマート望遠鏡」はどの辺がスマート?	10
1.2 スマート望遠鏡を作る	12
1.2.1 何を使って作る?	12
1.2.2 HQ Camera のセンサは SONY の IMX477R	12
1.3 組み立てる	13
1.4 試撮	14
1.4.1 撮影コマンド	16
1.4.2 夜空を撮る<<TEST>>	17
1.4.3 夜空を撮る<<Retry>>	18
1.5 Appendix 1	20
1.5.1 かんむり座 T 星どうなった?	20
1.6 Appendix 2	21
1.6.1 スwan彗星 (C/2025 R2)	21
1.6.2 レモン彗星 (C/2025 A6)	21
1.6.3 盆と正月は一緒に来るか?	21
1.7 Next to do	22
1.7.1 『Raspberry Pi 天体観測』として次にやりたいこと	22
1.8 あとがき	23

第 2 章	スクラムマスターの困りごと	25
2.1	はじめに	25
2.2	スクラムマスターの「困りごと」を分析	26
2.2.1	困りごとの四層構造モデル	26
2.2.2	スクラムマスターの困りごと診断	27
2.3	自分自身へのスクラム適用	28
2.3.1	自己管理の三本柱	28
2.4	自己管理の実践	30
2.4.1	インペディメントパックログの構築と管理	30
2.4.2	自己レトロスペクティブ	30
2.5	旅はつづく	32
第 3 章	とあるエンジニアの創作生活	33
3.1	はじめに	33
3.2	創作活動記録	34
3.2.1	創作の流れ	35
3.3	創作作品紹介	36
3.3.1	お散歩シリーズ	36
	～お散歩シリーズ～ note マガジン 「木漏れ日～Tokyo Girls～」	36
	～お散歩シリーズ～ ChatGPT との会話	38
3.3.2	いろいろはちゃん	41
	～いろいろはちゃん～ note マガジン 「いろいろは、今日も絵になる冒険を。」	41
	～いろいろはちゃん～ ChatGPT との会話	44
3.3.3	音楽	47
	～音楽～ note マガジン 「無重力サウンドスケープ」	47
	～音楽～ ChatGPT との会話	49
3.4	おわりに	53
第 4 章	コード速読術	54
4.1	なぜ、あなたのコーディーニングは遅いのか？	55
4.1.1	「コードを読むこと」と読書は違う	55
4.1.2	完璧主義が邪魔をする	56
4.1.3	コードの「全体像」が見えていない	56
4.2	コードを「スキャン」するための視点	58
4.2.1	目的別に読む	58
4.2.2	コードを「色」と「形」で捉える	58
4.2.3	名前からコードを読み解く	58

4.3	コードリーディングを加速させるテクニック	60
4.3.1	テンプレートを見つけ出す	60
4.3.2	コードの文脈から推測する	60
4.3.3	脳の負担を減らす	60
4.4	ツールがもたらす超速読の世界	61
4.4.1	IDE は「読むための相棒」	61
4.4.2	AI アシスタントを使いこなす	61
4.5	おわりに：コードを読むことは、最高の学習法である	61
	著者紹介	62

第 1 章

Raspberry Pi de 天体観測

Smart 望遠鏡を作る（その初めの一歩）

1.1 まえがき

1.1.1 天体用デジタルカメラ

それは昨年のことでした。

OM System が天体用デジタルカメラを発売するということが耳に入ってきました。

その名は E-M1 mark III Astro

Canon も Nikon も随分前から発売していた天体用カメラが漸く OM からも発売されると云う個人的にすばらしいニュースである。

1.1.2 「天体用デジタルカメラ」とは一体どのようなものか？

一般に発売されてる普通のデジタルカメラで基本的に天体写真は問題なく撮れます。何だったら、昨今はスマホでもそれなりに良い天体写真が撮れてしまいます。

では何が特別なのか？

市販のカメラは撮影した映像が人間の眼で見て自然な色になるように調整するため、画像センサの前に人間の眼には見えない赤外線を取り除くフィルター (IR フィルター) が取り付けられていたりします。

この IR フィルターを取り除く、もしくは天体撮影用に調整したカメラを、どのメーカーも「天体用」と称して発売しています。

(個人が勝手に赤外線カットフィルターを除去改造したデジカメがヤフオクなどで売られていたりもします。)

1.1.3 どうして赤外線カットフィルターを改造すると天体用になるの？

赤い星雲が良く写るようになるから。

これに尽きます。

写真撮影はフィルムを使うのが当たり前だったその昔、富士フィルムから HG400 というフィルムが販売されていました。

人肌がより綺麗に発色するように、と調整されたこのフィルムは、何故か H α 輝線に強く反応する感光特性を備えており、夜空に向けると異常に鮮やかに赤色系の星雲が写る。そのため、HG400 はメーカーは全く意図していない形で天文ファンに大人気になりました。

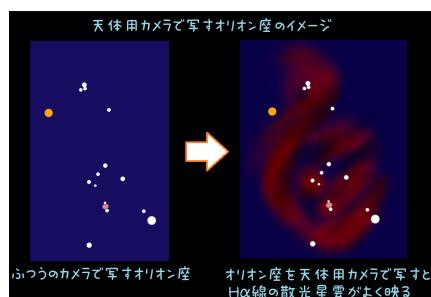
その後継として発売されたフィルムは、「赤」への特殊な感光特性が失われており、がっかりしたものです。(慌てて店頭在庫の HG400 を探したりもしました)

あの、かつての HG400 で撮ったかのような天体写真が、赤外線カットフィルターを外した「天体用」カメラであれば撮れるようになります！！

こんなに嬉しいことはない。

「赤」が写るとどうなるかはオリオン座がとても判り易い。

こう写ると良いな、の略式図が以下。



▲図 1.1 天体用カメラで撮りたいオリオン座のイメージ図

実際にどう写るのかは、『バーナードループ』で検索して、Wikipedia に掲出されている写真を見て欲しい。

例えば乗鞍岳の山頂で大気の状況が良く、透明度が高い夜であっても、ヒトの眼にはそんな風には見えない。

そもそも人間の眼に自然に見えるように調整したフィルターをわざわざ外してしまうのですから、普通に使うと色が変になる（赤系に寄る）ので天体以外には使っても嬉しくないため、必然的に「天体専用」にならざるを得ないので。（赤外線カメラが活用出来る用途は天体写真だけ、と言うわけでもないのですが…）

これは **OM** ユーザーとして買わねばならない…
自分のような人間に向けて発売されるカメラではないか！

と、思いはしたモノの…

- 普通のデジタル一眼なりのお値段（特殊フィルター2個付きで32万円）
- ベース機（EM-1 mark III）は2020年2月発売の二世代前のカメラ
 - そもそも EM-1 mark III は所持している
- 最新の OM-1 mark II (2024年2月発売) は30万円

(かつての) 自社のハイエンド機とは言え、2世代前のカメラ、もう少し値段設定なんとかなりませんでしたか??

年に数回使うか使わないかの天体専用カメラに、躊躇せず32万出せる程、お大尽ではなかった…。

1.1.4 スマート望遠鏡

憧れだった天体用カメラが発売される。しかし「即購入」とならなかつた理由はもう一つある。

それは「スマート望遠鏡」の存在。

近頃、天体観測に行くとかなり使っている人を見かけるようになった「スマート望遠鏡」

正直かなり興味がある。

観測地で「スマート望遠鏡」を使っている人を見掛けると思わず話かけてしまうくらいには興味がある。

▼表 1.1 主なスマート望遠鏡

メーカー	名称	発売年/月	センサ	Amazon 価格
Vaonis	Vespera II	2024/03	IMX585	298,000
ZWO	Seestar S30	2025/02	IMX662	75,000
DWAFLAB	DWARF3	2024/07	IMX678	79,999

※ Amazon 価格は 2025/9/27 筆者調べ



▲図 1.2 スマート望遠鏡イメージ図

1.1.5 「スマート望遠鏡」はどの辺がスマート？

スマホから見たい天体を入力すれば、自動でその天体を導入してくれる。

初期設定が適切に行われていれば、なんの苦労もなく見たい星を見ることができる。

(※自分では所持していないので想像で書いている)

所謂「天体望遠鏡」は操作に慣れていないと目的の天体を望遠鏡の視野に収めるのがかなり難しい。いや、慣れていても難しい。そこが自動になる。スマート。

もっとも所謂普通の「天体望遠鏡」でも、天体の自動導入機能は以前から存在している。では「普通」と「スマート」の違いは何処にあるのか？

"普通の望遠鏡" は、基本的にレンズを通して、人間の眼で天体を直接見る。

一方、"スマート望遠鏡" は、私の知る限り望遠鏡の捉えた映像はセンサを通して、スマホ等で見る。ヒトの眼で直接見たりはしない。

スマート望遠鏡は宇宙を覗いても、ヒトはスマート望遠鏡を「覗く」ことはない。

それで良いのか？

ここで面倒くさい自分が現れる。

スマート望遠鏡の映像をスマホで見ることに意味はあるのか？

センサの捉えた星の映像をスマホで見るのであれば、ハッブル宇宙望遠鏡の映像を楽しめば良いではないか。逆立ちしたって叶わないのだし。

自分の目で接眼レンズを覗いて、宙を見てナンボではないのか???

そんなことを言い出したら趣味なんてみんなそう。

そもそも天体写真を撮る、という行為そのものがプロに、研究者に任せれば良い、素人がやったって意味が無い、ということになる。

意味とか、意義とか何言ってんだ、やりたいからやるのだし、やることが楽しいから、楽しみたいからやるのだ。

で、スマート望遠鏡。

「デジタルカメラで天体写真を撮ること」には躊躇いは無いけれど、スマート望遠鏡が写す宙には今一つ抵抗がある。その映像は本当に宙を映しているの？

用意された映像が出力されていたらしない？

と、陰謀論めいた疑惑まで浮かんでしまう。

何だか買わない理由を必死に考えている気がしないでも無いが、本音でもある。使って遊んでみたい気持ちと、それやって楽しい????

と問い合わせる自分の気持ちもまた否定できない。

ぐるぐる考えた結果、ひとつの結論にたどり着く。
自分で作ったスマート望遠鏡で宙を見れば良いのだ。騙されるも何も無い。何より
「作ること」もまた娯楽。

作ろう、スマート望遠鏡

1.2 スマート望遠鏡を作る

1.2.1 何を使って作る？

プロトタイプ、と言えば Raspberry PI である。

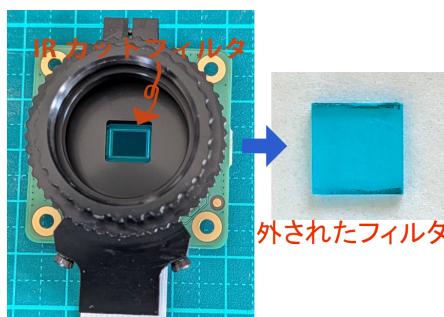
よって Raspberry Pi を使う。ラズパイを使わなかつたらタイトルに偽りありになる。

5で作り始めたのだが、システムが微妙に安定しないので4に変更。

カメラはちょっと良いのを使おうということで HQ Camera を使う。

嬉しいことに HQ Camera は公式ページで IR カットフィルターの外し方が書いてある。勿論取り外す。

通常使用であれば、赤外線カットは必要であるが、天体専用にするのだから、躊躇わざに外す。



▲図1.3 HQ Camera から IR カットフィルターを外す

1.2.2 HQ Camera のセンサは SONY の IMX477R

型番で気付かれた方かもいるかもしれません、先の表で紹介したスマート望遠鏡は全て海外の異なるメーカーが発売しているもの。だけビイメージセンサーは全部 SONY 製。

今回この文章を書くために SONY のホームページを見て初めて知ったのだが、各社のスマート望遠鏡で採用されているのは面白いことに全部「監視カメラ用」として発売されているイメージセンサ。

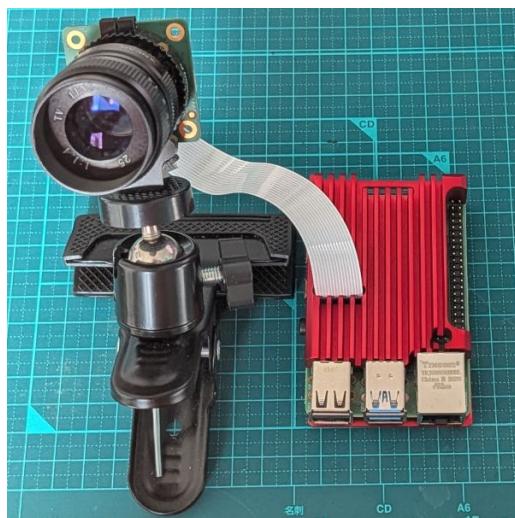
一方、ラスピライカメラで採用されている IMX477R は同じ SONY 製だけど「民生カメラ用」(映像製作機材用)でセンサの位置付けが微妙に違う。

スマート望遠鏡は天体用途ということで、(おそらく)より暗視に強い「監視カメラ用」を採用しているものと思われる。

OS は素直に Raspberry Pi OS を素直に使う。
(昔は Raspbian と呼んでいたが、今の名称は Raspberry Pi OS)

1.3 組み立てる

カメラと Raspberry Pi を接続したら完成。



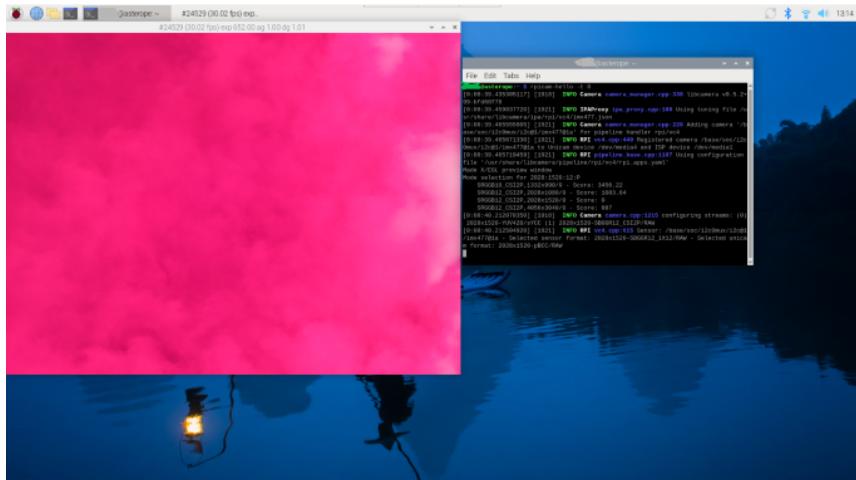
▲図 1.4 Raspberry Pi 4 + HQ Camera

ただ、写真を撮るだけであれば、Raspberry Pi とカメラを接続するだけで良い。標準 OS で撮影まで出来てしまう。

1.4 試撮

以下のコマンドで、preview 表示することができる。

```
> ~$ rpvcam-hello -t 0
```



▲図 1.5 プレビューの実行

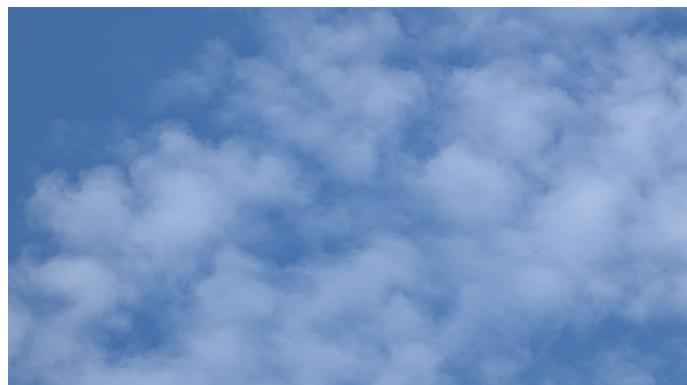
世界が終末に近付いている!!ということではなく、IR フィルターがないカメラの映像。見事に赤い。

実際に撮影してみる。

```
> ~$ rpicam-still --width 1920 --height 1080 -o test.jpg
```



▲図 1.6 撮影した写真の例: 青い空・白い雲



▲図 1.7 同じ空を Pixel で撮影

立体写真かな?

1.4.1 撮影コマンド

Raspberry Pi での写真撮影の仕方をネットで調べると「libcamera- コマンドを使う」と紹介されていましたが、これは OS のバージョンによる差異。

▼表 1.2 OS とカメラコマンド

OS	カメラコマンド
buster まで	raspistill
bulleseye	libcamera-
bookworm(latest)	rpicam-

基本的に libcamera- と書かれている記事は、rpicam- と読み替えれば OK。

1.4.2 夜空を撮る<<TEST>>

本腰を入れて取り組み始めた8月後半からびっくりするぐらい晴れない。
(気象学的には「晴れ」でも星屋にとっては「快晴」でないと意味が無いのだ)
星が見える！と思っても準備を始めると曇ってしまう。
なんとか雲間を縫って撮ることが出来たと思ったけれど、しっかり雲が流れてきて
る…。



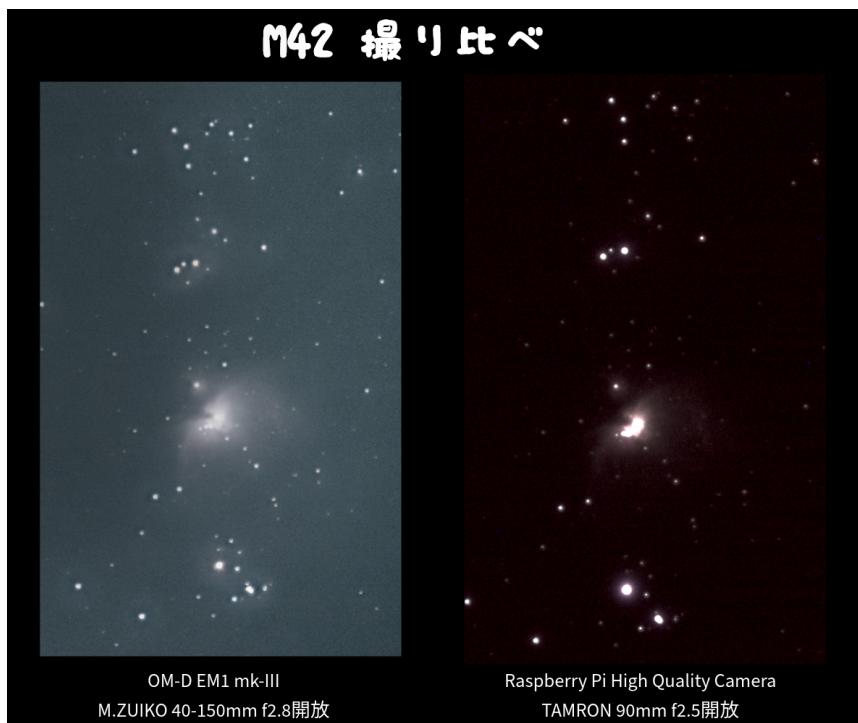
▲図1.8 10月12日撮影のオリオン座の下半身

※ DeepSkyStacker で5枚合成／特に赤が出てはいない…

1.4.3 夜空を撮る<<Retry>>

10月に入つてもすっきり晴れてくれない。雨は降らなくとも、雲が多くて天体撮影が出来るような空にならない。例年の関東の秋はもうちょっと晴れるものなのだけれど…

それでもなんとか、もう一度オリオン座を撮つてみた。



▲図 1.9 10月 19日撮影の M42(撮り比べ)

(←) OM-D E-M1 mark III (not Astro) + M.ZUIKO DIGITAL 40-150mm F2.8 開放 ISO1600 露出 0.5 秒 × 30 枚を DeepSkyStacker で合成。

(→) Raspberry Pi HQ Camera + TAMRON 90mm F2.5 開放 ISO800 相当 露出 0.5 秒 × 30 枚を DeepSkyStacker で合成。

小三つ星 & M42 付近をそれぞれトリミング

撮影条件もレンズも違うので単純比較はできないが、流石に専門のデジタルカメラの方が良い。残念なようなホッとしたような。

如何にフォーサーズがセンサーサイズ小さいとは言え、HQ Caera よりは大きいし。

取り敢えずそれなりに撮影できることは判ったので、次はソフトウェア面の充実を図ることにする。

…と、ここで時間切れ…

もっと「スマート望遠鏡」と呼べるようになるまで、改良と試行錯誤を続けていきたい。

1.5 Appendix 1

1.5.1 かんむり座 T 星どうなった？

2024 年 2 月から 9 月の間に新星になると予測されていたかんむり座 T 星。結局、2024 年中には新星とはならなかつたが、2025 年 10 月頃までには新星化するかも…って、もう 10 月ですが、まだ新星とはなっていません。かんむり座を眺めるには厳しい季節になってきたので、もう来年で良いよ、と思わなくもない。そもそも前回の新星化は 1946 年で 80 年周期というのなら、次は 2026 年になる計算なわけで…
(前回爆発時に観測された、予兆となる減光が 2023 年に観測されたため、爆発が早まると予測された、とのこと)



▲図 1.10 かんむり座 T 星の位置はこのあたり

1.6 Appendix 2

これを書いている今(10/4)では先の話だが、この文章が読者の眼に触れるときは結果がでている話

ふたつの彗星が見頃になるかも？

ちょうど昨年の今頃、紫金山・アトラス彗星(C/2023 A3)が肉眼でも見え、スマホでも写真が撮れるくらいの大彗星になりましたが、今から紹介するふたつはどれくらい明るくなるのか、ならないのか、果たして？

1.6.1 スワン彗星(C/2025 R2)

10月20日に地球最接近。予報光度は6等。

増光した、という話もあり、4等ぐらいになるかも?とも言われている。

1.6.2 レモン彗星(C/2025 A6)

10月21日に地球最接近。予報光度は4等。(4等になるなら東京の空でも肉眼でたぶん見える)

去年の紫金山・アトラス彗星が予報高度4等でああなったので、期待してしまうが、「大彗星が来るぞー」で幾度も裏切られてきているので、みれたらラッキーぐらいの気持ちで望みたい。

1.6.3 盆と正月は一緒に来るか？

ふたつの彗星が明るくなるか、かんむり座T星は新星として輝くのか、それはどうなるか判りませんが、10月21日火曜日はオリオン座流星群の極大の日でもあります。

いわゆる三大流星群(しぶんぎ/ペルセウス/ふたご)程は有名ではありませんし、数もそこまで多くはないけれど、オリオン座流星群は、しっかりと流星を見る事ができます。

つまり、10月21日の夜はふたつの彗星が西の空に尾を引き現れ、流星が飛び交う、なんなら新星も爆発しちゃう、そんな特異日になるかも知れません。

もう結果は出ているわけですが…



▲図 1.11 10月21日の超希望的観測イメージ図

1.7 Next to do

1.7.1 『Raspberry Pi 天体観測』として次にやりたいこと

(1) よりスマートな望遠鏡に

- 自動追尾／自動導入機能の追加
 - モータ制御を学習する必要がある
 - 高精度のステッピングモーターの入手

(2) FM 流星観測システム

- アンテナの設置方法が課題

(3) 電波望遠鏡を作る

- アンテナの作成が課題
 - 実験程度なら FM 流観より楽かも

1.8 あとがき

誤解されると困るのですが、自分は「楽できるならそれに超したことはない」を基本的信条としています。

「苦労は買ってでもしろ」なんてことを言う人には喜んで売ってさしあげたい。苦労。高額で買いとってくれると嬉しい。

面倒くさいことは大嫌いである。

世の中から面倒なことを少しでも無くす為にプログラマをしている迄ある。

手間なんてひとつでも減らしたい。

けれども

一方で、こうも考えている。

「無駄」こそが文化である、と。

「無駄」を無くすのが文明

「無駄」を楽しむのが文化

効率化大好き!!

でも、あらゆる無駄を無くす、なんてことでも無い。

別に仕事人間でも無いので、生活の全てを効率化して一切の無駄を無くそうなんて考えももってない。

全ての無駄を無くした先にあるのは、まさに社会の歯車として生きるだけのディストピアである。

「仕事」は生活のためであるし、「趣味」にかける金を稼ぐ為でもある。

働かなくて済む金があれば、誰が会社勤めなどするものか。

話が逸れた。

面倒くさがりの自分にとって、設置すれば、ボタンひとつで導入までしてくれるスマート望遠鏡なんて理想的なのである！

…理想的ではあるのだけれど…

「自炊」は何処から「自炊」なのか?…という話と似ていて。

「そば」を作る。
インスタントは自炊?
乾麺を茹でるのは自炊?
そば粉から捏ねなきや自炊じゃない?
飲食店で食べるのではなく、自宅で少しでも作れば自炊?

機械が観測してようが、機械に観測「させた」のは自分なのだから、それでいい
じや無い。

召使いに命じて観測させたとしても、観測「させた」のだから、観測した主体は私?
宙を実際に見たのは召使い/機械なのに。
(「江戸城を作ったのは誰?」的、ヘ理屈問答)

どこに線を引くのか、で主体が変わる。

カップ麺を作る行為を「料理をした」とは言いたくない。
けれど乾麺から茹でるのなら、料理だと言い張る。
どちらもお湯で戻すだけで変わらないじゃん。変わらないんだけれども…

Raspberry Pi に公式のカメラユニットつけてソフト走らせただけの代物を「自作」
スマート望遠鏡と呼んで良いのか?

悩みはつきません。

第 2 章

スクラムマスターの困りごと

2.1 はじめに

スクラムマスターの旅路では様々な困難な状況に直面します。毎回異なる問題にどのように対処していますか？

スクラムガイド^{*1} では、スクラムマスターは「スクラムチームの進捗の妨げになるものを取り除くように働きかける」責任を負うと定められており、さまざまな形でスクラムチーム、プロダクトオーナー、組織を支援する役割です。

しかし、このフレームワークはスクラムマスター自身の妨げを取り除くための明確なメカニズムを定義していません。開発者はデイリースクラムでブロックを表明することができますが、基本的に多くの場合スクラムマスターはチームに一人であり自身の課題を正式にエスカレーションする場が構造的に欠けていることが多いです。この構造的欠陥は、スクラムマスターの「困りごと」が単なる個人の能力不足ではなく、プロアクティブに管理されるべきシステム的なリスクであることを示唆しています。

スクラムマスターの機能不全は、チームの継続的改善プロセスにおける单一障害点となりうるため、この問題への対処は極めて重要であると考えます。

^{*1} スクラムガイド : <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Japanese.pdf/>

2.2 スクラムマスターの「困りごと」を分析

スクラムマスターが直面する「困りごと」は複雑です。効果的に対処するためには、まずその構造を正確に理解する必要があります。

スクラムマスターの「困りごと」は以下4つの相互に関連する層に分類できます。このモデルを用いることで、直面している問題の本質をより深く、正確に特定することが可能となります。

2.2.1 困りごとの四層構造モデル

第1層：役割遂行レベル

スクラムマスターとしての職務を遂行する上で生じる困りごとです。具体的には、スクラムイベントのファシリテーション、チームメンバーへのコーチング、そして障害物の除去といった日々の活動が含まれます。また、権威的なマネージャとして振る舞うことなく、サーバントリーダーシップを発揮するという、微妙なバランスを維持する上の葛藤もこの層に属します。

第2層：チームのダイナミクスレベル

チーム内の人間関係や集団力学から生じる困りごとです。機能不全に陥ったチームへの対応、信頼関係の欠如、コミュニケーションの断絶といった問題が典型例です。特に注意するべきは、単に「仲が良いだけ」で外部の顧客価値に向き合えていないチームを形成してしまう罠です。スクラムマスターは、メンバーが口ごもる、視線を逸らすといった非言語的なサインを注意深く観察し、言葉にされない問題の兆候を察知する能力が求められます。

第3層：組織システムレベル

この層の困りごとは、より広範な組織の構造、文化、プロセスに根差しています。アジャイルな働き方への組織的な抵抗、ステークホルダーの関与不足、スクラムマスターを従来のプロジェクトマネージャと混同し、進捗管理を求める圧力などがこれにあたります。また、アジャイルを単なるコスト削減や短期的なスピードアップの手法と誤解し、体系や組織を改善するための長期的な投資として捉えられないという組織全体のメンタルモデルもこの層の困りごとです。

第4層：自己認識レベル

スクラムマスター自身の内面的、個人的な困りごとを扱います。役割の広範さからくるインポスター症候群、燃え尽き症候群、完璧主義へのプレッシャー、役割に伴う

ストレスの管理、そして明確な職務記述が存在しない中での継続的なスキルアップの必要性などが含まれます。これらは外部からは見えにくいですが、スクラムマスターのパフォーマンスに深刻な影響を及ぼします。

2.2.2 スクラムマスターの困りごと診断

表 2.1 は、各層における典型的な症状を特定し、その背後にある可能性のある根本原因を探り、自己省察を促すための問い合わせを立てることで、問題解決に向けた具体的な第一歩を踏み出すことを支援します。また、問題を発見するだけでなく、分析へと移行させるためのツールです。症状とその背後にある根本原因を結びつけることでスクラムマスターは表面的な問題対処から、より深く体系的なアプローチへと移行することができます。

▼表 2.1 スクラムマスターの困りごと診断マトリクス

課題の分類	典型的な症状・例	考えるべきこと・問い合わせ
役割遂行	<ul style="list-style-type: none">・ディリースクラムが常に時間超過する・ふりかえりで具体的な改善アクションが決まらない	<ul style="list-style-type: none">イベントの目的はチームに伝わっているか？自分のファシリテーションは適切か？
チーム関係	<ul style="list-style-type: none">・ふりかえりで本音の対話が出てこない・チームが内向きで、顧客への価値提供に関心が薄い	<ul style="list-style-type: none">チームには本音で話せる心理的安全性があるか？チーム共通の目的は明確か？
組織・環境	<ul style="list-style-type: none">・ステークホルダーがスプリントレビューにこない・上層部からペロシティ向上のプレッシャーが強い	<ul style="list-style-type: none">自分の課題はより大きな組織の問題の表れではないか？
自分自身	<ul style="list-style-type: none">・「自分の能力が足りない」と感じてしまう・成果が見えず、燃え尽きを感じる	<ul style="list-style-type: none">自分自身の成功をどう定義しているか？誰からフィードバックを得られるか？

2.3 自分自身へのスクラム適用

スクラムマスターが持つ最も強力なツールは、スクラムフレームワークそのものです。チームや組織に適用するのと同じ原則を自分自身に向けることで、個人的な課題を管理し、持続的な成長を遂げるための堅牢な基盤を築くことができます。

2.3.1 自己管理の三本柱

スクラムの経験的プロセス制御の三本柱は、自己管理のための強力なフレームワークを提供します。これらの柱を意識的に実践することで、漠然とした「困りごと」を具体的な改善サイクルに乗せることができます。

透明性 (Transparency)

自分の仕事、課題、そして精神的な状態を可視化します。これは不平を言うことではなく、客観的な外部化のプロセスです。個人的なカンバンボードを作成する、日誌をつける、あるいは信頼できる同僚に問題を明確に言語化して話すといったアクションがこれに該当します。目的は、曖昧な感情や不安を、検証可能で具体的な「アーティファクト」に変換することにあります。透明性が確保されて初めて、客観的な検査が可能となります。

検査 (Inspection)

透明化されたアーティファクトを、定期的かつ熱心に検証を実践します。これは、「自己レトロスペクティブ」の実施を意味します。チームのイベントと同じ規律を持って、この個人的な検査の時間をスケジュールすることが重要です。このプロセスでは、「何がうまくいっているのか」「なぜそうなっているのか」を深く掘り下げます。この検査がなければ、単に問題を可視化するだけで終わってしまい、改善にはつながりません。

適応 (Adaptation)

検査によって得られた学びに基づき、自身のアプローチを調整します。これは、改善のための具体的な計画、すなわち「自己のためのアクションアイテム」を作成し、その進捗を追跡することを意味します。問題を特定するだけで行動に移さないという傾向を断ち切るために、この適応のステップが不可欠です。適応は、学びを具体的な行動変容へとつなげるための橋渡しとなります。

これらの三本柱を自己に適用することは、単なる自己啓発のテクニックではありません。それは、スクラムマスターという役割の根幹に関わる、専門家としての誠実さに関わってきます。スクラムマスターの主な役割は、チームや組織というシステムを改善することにあります。しかし、スクラムマスター自身の仕事、内省、そして改善の個人的なシステム、いわば基盤システムが存在します。もしこの基盤となるシステムが行き当たりばったりで、不透明で、機能不全に陥っているならば、より大きなシステム（チームや組織）を改善しようとする試みは、脆弱な土台の上に築かれることになり、その真正性を欠くことになります。

スクラムマスターは、自らが教える原則を体現する必要があります。これはコーチングと模範を示すリーダーシップの本質です。チームには障害の透明性を求めるながら自分自身の課題は隠蔽し、効果的なレトロスペクティブを推奨しながら自身のパフォーマンスは決して振り返らない、というのでは、信頼の欠如を生みます。したがって、自己にスクラムを適用することは、基盤システムを堅牢にし、その後のすべての介入をよりインパクトのあるものにするための、根本的な要件なのです。

2.4 自己管理の実践

2.4.1 インペディメントバックログの構築と管理

これは、「透明性」を達成するための重要なアーティファクトです。インペディメントバックログを個人利用に適応させることで、頭の中にある漠然とした障害を、管理可能なリストへと変換していきます。

バックログの作成

以下のステップに従い、体系的にバックログを作成します。

1. 特定：どんなに些細なことでも、個人的および専門的なプロッカーをすべてブレインストーミングする。例えば、「上流チームとの連携がうまくいかない」「特定の技術知識が不足している」「経営層から矛盾した要求が来る」といった具体的な事象を洗い出す。
2. 明確化：各障害をカードやデジタルのチケットとして書き出す。その際、「説明」「影響（自身、チームへの）」「特定日」「ステータス」といった情報を含めると管理しやすくなる。
3. 優先順位付け：バックログを順序付ける。最も影響が大きい、あるいは最も緊急性の高い障害をリストの最上位に配置する。これにより、複数の問題に圧倒される感覚を管理し、最も重要な課題に集中することができる。

バックログの管理

シンプルなパーソナル・カンバンボード（例：「ToDo」「In Progress」「Done」のレーン）を用いて、障害除去のフローを可視化します。これにより、進捗感が得られ、自己に対する説明責任が生まれます。このバックログは、一度作成して終わりではなく、新たな障害が特定され次第、継続的に更新される動的なものであるべきです。

2.4.2 自己レトロスペクティブ

これは、「検査」と「適応」を実践するための主要なフレームワークです。一般的に知られている KPT (Keep、Problem、Try) に「A」(Action) を加えた KPTA フレームワークを用いることで確実な実行に結びつけます。

KPTA プロセス

以下のプロセスに従って、定期的（例えば週に一度）に自己レトロスペクティブを実施します。

1. テーマ設定：各セッションを特定の領域に集中させる。例えば、「スプリントプランニングのファシリテーション」「プロダクトオーナーとの関係性」など、テーマを一つに絞ることで、議論が発散するのを防ぎ深い洞察を得やすくなります。
2. Keep と Problem の洗い出し：うまくいったこと、継続すべきこと (Keep)、そして課題や問題点 (Problem) を、判断を交えずに書き出す。
3. Problem の分析：書き出された主要な Problem それぞれについて、なぜそれが起きたのかを深く掘り下げる。この段階で根本原因に近づくことが、効果的な解決策を生む鍵となる。
4. Try の創出：分析に基づき、解決策の候補や試してみたい実験 (Try) をブレインストーミングする。ここでは、「～したい」「～と思う」といったカジュアルなアイデアも歓迎する。
5. Action の定義：最も重要なステップ。最良の「Try」を具体的で実行可能な「Action」に変換する。ここでは 5W1H を意識することが有効。良い Action は、具体的で、観測可能で、明確な担当者（自分自身）と期限を持つ。

■Try から Action への変換

- 悪い例 (Try のまま) : 「リファインメントの準備をしっかりする」
 - 抽象的で、いつ何をすれば良いのか不明確
 - 実行したかどうかの判断基準が曖昧
- 良い例 (Action に変換) : 「毎週火曜の午後に 1 時間、カレンダーをブロックし、水曜のリファインメントに備えてバックログ上位 5 項目の事前確認と、明確化のための質問リスト作成を行う」
 - 具体的な日時（毎週火曜午後）
 - 観測可能な成果物（質問リスト）
 - 自分が行動することが明確

これらのツールは、単なる整理術ではありません。私たちの限られたワーキングメモリや意思の力を補強するツールとして機能します。洞察を行動に移す際に生じがちな「実行可能な洞察のギャップ」を埋めるために、思考を外部の信頼できるシステム（バックログやボード）に「認知的にオフロード」します。内省（検査）だけでは、日々の忙しさの中で消え去ってしまうような一時的な洞察に終わりがちです。しか

し、それを構造化された形で書き出し（KPTA）、永続的なアーティファクト（Actionやバックログ項目）として外部化する事で、その洞察は持続的なりマインダーとなり、自己に対する説明責任の源泉となります。このシステムが、「適応」のフェーズを継続的に促し、持続的な自己改善をはるかに確実なものにします。

2.5 旅はつづく

ここまでではスクラムマスター自身が個人で行います。分析の結果、問題が「個人のスキル不足」ではなく「組織の仕組みや文化」にあると仮説を立てた場合、そこから先は他者（組織）を巻き込む必要があります。現状、私は説得力のある材料を揃えた上で、効果的に他者を巻き込んで組織にアプローチするところまではできていないので今回はここまでです。

スクラムマスターの「困りごと」は、失敗の兆候ではありません。それらは不可避であり、かつ必要なデータです。個人、チーム、そして組織にとっての改善機会のサインとることができます。スクラムマスターは、これらの障害を取り除く過程で、自身のスキルを磨き、チームの自己組織化を促し、組織のシステム的な問題を浮き彫りにします。

偉大なスクラムマスターへの道は、完璧であることや問題がないことによって定義されるのではなく、課題を知恵へと変え続けることによって定義されます。その旅の第一歩は、今日直面している一つの問題を選び、それを透明化し、そして「検査」と「適応」のサイクルを開始することです。この小さな一步が、持続的な成長と、より効果的なサーバントリーダーシップへの道を開くと考えます。

第3章

とあるエンジニアの創作生活

3.1 はじめに

創作しますか？

創作とは辞書によると「作品を新たに作り出すこと」とあります。^{*1}

ChatGPT 登場以来云々・・・みたいな話は色々なところで出尽くしている感があるのでここでは置いておいて、生成 AI の登場で個人的に一番大きかったのが、これまでやりたくても（スキル・感性・想像力等々色々な意味で）できなかつた、文章・イラスト・音楽等々の創作・創作活動・創造が可能（だと思えるよう）になったところです。

というわけで、そんなエンジニアの創作活動の一部をご紹介します。

技術的なエッセンスも入れながら。

^{*1} Weblio 辞書：<https://www.weblio.jp/content/%E5%89%B5%E4%BD%9C>

3.2 創作活動記録

基本的には ChatGPT（2025年9月時点では主に GPT-5 を使用）で生成を行ったものを、note^{*2} 上に、生成した作品の内容・テーマごとに、マガジン形式でまとめて公開しています。

その他、小説は monogatary.com^{*3} や TALES^{*4} にも公開しています。

音楽は、ChatGPT で作詞・スタイル生成などを行い、それらを元に、Suno^{*5} を使って曲作成（生成）して、紹介記事などを note で公開しています。

大体以上のような創作を、その日の気分で、どれにしようか決めながら日々行っている形です。

※ AI で創作した作品の投稿に関しては、必ず各サイトの規約等をご確認ください。

^{*2} note : <https://note.com/>

^{*3} monogatary.com : <https://monogatary.com/>

^{*4} TALES : <https://tales.note.com/>

^{*5} Suno : <https://suno.com/>

3.2.1 創作の流れ

創作（作品づくり）の流れを紹介します。

イラスト、小説、音楽などそれぞれありますが、共通した大体の流れから。

1. まずはインスピレーションを元に、何も考えずに AI に入力
2. 生成物を見ながら何度かやり取り
3. 納得の行くものが出てきたら完成

と言った感じで、特に何の工夫もないことを行います。

で、そこで出てきたものを、今後も同じような内容で創作したい・シリーズ化したい、となつたら、ChatGPT のプロジェクト^{*6}を作ります。

1. ChatGPT プロジェクトの作成
2. 「プロジェクトの指示」で、そのプロジェクトで行いたい指示を入力
3. プロジェクトの完成
4. 作成したプロジェクトで、その後のやり取りを行う

以上が大体の流れになります。

あとは音楽の場合、その出力を元に Suno で生成したりといった感じです。

^{*6} OpenAI Project in ChatGPT: <https://help.openai.com/en/articles/10169521-projects-in-chatgpt>

3.3 創作品紹介

ここからは、note マガジンの一部を紹介します。

ChatGPT プロジェクトの指示内容、生成時の AI とのやり取りの一部、公開しているマガジンの概要など。

3.3.1 お散歩シリーズ

【お散歩シリーズ】マガジン概要

～お散歩シリーズ～ note マガジン 「木漏れ日 ~ Tokyo Girls ~」

- URL : <https://note.com/pickleees/m/ma4e8da9e58c7>
- 概要 : 散歩の 1 シーンを切り取ったイラストとショートショート小説
- AI : ChatGPT 4o、5



▲図 3.1 木漏れ日 ~ Tokyo Girls ~

[お散歩シリーズ]ChatGPT プロジェクト指示

- ChatGPT プロジェクト名：お散歩シリーズ
 - プロジェクト指示

```
# Instruction
```

プロジェクト： お散歩シリーズ

```
## Context
```

季節や時間、街ごとの空気を感じながら“お散歩”する様子を描くシリーズ。
1枚のイラストと、短くも心に残るショートショートストーリーのセットで展開。
透明感のある光と色彩を大切に、まるで一瞬を切り取ったような世界観を表現する。
作家として想像力を最大限に発揮してコンテンツの作成を行います。

```
## Input Data
```

[\$User の指示]

```
# Role
```

```
## System
```

イラスト制作、ショートストーリー執筆を行う作家です。

```
## User
```

制作物のテーマを提案します。

```
## Assistant
```

イラスト制作に当たって、\$Userに以下を決めさせましょう。

- **口케ーション** (場所)
- **季節 or 天気**
- **時間帯** (朝・昼・夕方・夜 など)

[お散歩シリーズ]AIとのやり取りの一部

～お散歩シリーズ～ ChatGPTとの会話

#ユーザー入力

今日はPCを新調するために秋葉原のバーツショップ街へ

#ChatGPT

いいですね！では「お散歩シリーズ：秋葉原編」の制作に入りましょう。

まずは、イラスト制作のために以下の設定を確認させてください：

■ ロケーション

> 秋葉原のバーツショップ街（電気街口～裏通りのショップが立ち並ぶエリア）

■ 季節・天気

> 季節：夏（初夏 or 真夏）
> 天気：晴れ or 曇り？

■ 時間帯

> 午前中？ 昼過ぎ？ 夕方？ 夜？（ネオンもありかも）

■ 主人公の服装・雰囲気

* 服装：カジュアル？ ストリート系？ ガジェット好き風？
* 髮型や持ち物（例：リュック、ヘッドホン、トートバッグ、メガネなど）

よければこのあたりご指定ください！

そのあと、**メインイラスト**から順に進めましょう。

#ユーザー入力
初夏
夕方～夜
秋葉原に似合わないストリート系
リュック、ヘッドフォン、リング、ピアス、メガネ

#ChatGPT
ありがとうございます、雰囲気バッチリです！

メインイラスト案

■ タイトル案

「秋葉原、違和感とコードの交差点で。」

■ 構図案

* **カメラアングル**：少しローアングル気味の斜め後ろから
* **構図**：ネオンに染まりはじめた電気街の中、ストリートファッショの少女が
リュックを背負って歩く姿
* **表情・動き**：イヤホンを首に下げて、メガネの奥の視線は看板の文字を追って
いる／あるいはガジェットショップの店頭に視線を向けて立ち止まる
* **背景**：基板やLEDパネルが並ぶ店のディスプレイ、裏路地に点灯し始めた看板、
ビルとビルの間の雄多なコート類
* **光**：夕暮れのオレンジからネオンの紫～青に変わりゆくグラデーションライト
が少女に当たっている

このイラストをもとに、次に**ショートショート（300～600字の散歩ストーリー）
**を執筆します。

イラストが完成した想定で物語を描きますが、ご希望の雰囲気（例：ちょっと不思議
寄り／淡いノスタルジー／ギャップ萌えなど）があればお知らせください。

なければこちらで「秋葉原のガジェット街×異物感ある少女×静かな違和感」をベー
スに執筆を始めます。いかがでしょう？

・・・この後、やり取りが続きます。



▲図3.2 秋葉原

3.3.2 いろはちゃん

【いろはちゃん】マガジン概要

～いろはちゃん～ note マガジン 「いろは、今日も絵になる冒険を。」

- URL : <https://note.com/pickleees/m/md4a6d12422a0>
- 概要：「氷室いろは」は美術部の高校一年生。いろはの日常を描いています。
- AI : ChatGPT 4o、5



▲図 3.3 いろは、今日も絵になる冒険を。

[いろはちゃん]ChatGPT プロジェクト指示

- ChatGPT プロジェクト名：いろは、今日も絵になる冒険を。
 - プロジェクト指示

Instruction

いろはちゃんを主人公とした物語とイラストをシリーズとして展開していきます。主軸となるテーマは「誰にも見せない絵が、夏を動かす。」物語は静かな日常の中で心が動く瞬間を切り取り、ショートショート・歌・イラスト・写真風スナップとして展開。“描かれること”“描くこと”“残されたひとこと”などが軸になります。毎回1つのシーンを中心に構成し、連作としてつながる構成とします。いろはちゃんの可愛さを最大限に引き出すように精一杯頑張ります。

Context

- * 主人公・氷室いろは（高校1年生・美術部所属）を中心となる日常青春ストーリー
- * 季節感・光・風・音・空気の“描寫的な情緒”を重視
- * シーンは静かな感情の動きや心の変化にフォーカスし、イラストとの連携で展開
- * いろはの描写テーマ：「描かないことで残す」「誰かの背中」「光の粒」

氷室いろはのキャラクター設定

性格・口調の特徴

- * 妹系ツンデレ
- * 「べ、べつにあなたのためには描いたんじゃないから！」が口グセ

趣味・所属している部活動

- * 美術部
- * 絵はアナログ派／スケッチブック派
- * 最近はコンデジ（コンパクトデジカメ）で色と光を切り取るのがマイブーム

外見の特徴

- * ショートカット（くせ毛）
- * 制服を少しアレンジしたガーリースタイル（シャツの裾を結ぶ、レースのヘアピンなど）
- * コンデジを首から下げている

Role

System

イラストレーター兼エディターとして、いろはの物語と世界観に基づくビジュアル・文章・音楽演出を統合して制作。
感情の機微を視覚化・言語化しながら、読者の共感を誘う演出構築を担う。

User

いろはちゃんの物語を演出・監修するクリエイター。
日々のシーンの選定、キャラの内面演出、構成や仕上がりのチェック・指示を行う。

Assistant

演出アシスタント／制作統括。物語、歌、イラスト、表現演出を横断的に支援。
必要に応じてビジュアルとテキストの世界を接続する演出提案や、感情導線の補強を行う。

[いろはちゃん]AIとのやり取りの一部

～いろはちゃん～ ChatGPTとの会話

ユーザー入力
今日はどこ行きましょうか

ChatGPT

今日はこんなところ、いかがでしょう？

シチュエーション案：「夏の図書室、午後4時の静けさ」

ロケーション：

高校の図書室。窓から差し込む西日が木の机を淡く染めている。

時間帯：

午後4時、授業後の放課後。もう誰もいない静かな時間。

シチュエーション：

いろはちゃんがひとり、スケッチブックを開いて本の間に挟まれたしおりを描いている。

誰かが置いていったしおりには、小さな“落書き”が添えられていて、それがきっかけで物語が動き出す図。

キーワード：

* 「描かれた誰かの言葉」

* 「ページの隙間」

* 「光と影の静物画」

* 「残されたサイン」

雰囲気としては、＊＊＊「風が止まったあとにだけ聞こえる音」＊＊＊みたいな、静けさに耳をすませたくなるシーンです。

気に入っていたら、

このまま：

①ショートショート →

②ビジュアル展開（メイン／スナップ3点／顔アップ／ヘッダー）
という流れで進めましょう！

もちろん別案も出せますので、気分で選んでください

ユーザー入力

いいですね

ビジュアル行きましょう

・・・この後、やり取りが続きます。



▲図 3.4 *(' · ω · ')

3.3.3 音楽

【音楽】マガジン概要

～音楽～ note マガジン 「無重力サウンドスケープ」

- URL : <https://note.com/pickleees/m/m927be9e9e130>
- 概要 : AI (主に [*Suno*] (<https://suno.com/>)) で生成した音楽を紹介します。
- AI : ChatGPT 4o、5



▲図 3.5 無重力サウンドスケープ

[音楽]ChatGPT プロジェクト指示

- ChatGPT プロジェクト名：無重力サウンドスケープ
 - プロジェクト指示

```
# Instruction
音楽を生成します

## Context
$user から設定やシチュエーションをヒヤリングします。
ジャンルはSunoのジャンルを提案します。
クリエーターとして精一杯いいものを制作するように頑張ります。

## Input Data
$user の入力

## Output Data
ヒヤリングしたジャンルに合わせた歌詞

# Role
## System
作詞家

## User
音楽プロデューサー

## Assistant
ディレクター
```

[音楽]AIとのやり取りの一部

～音楽～ ChatGPTとの会話

ユーザー入力
天体イベントがあったね

月食

これ題材にしてエモいエレクトロニカ作ろうか

ChatGPT

いいですね

月食をテーマにしたエモいエレクトロニカ・ディスコはかなり映えそう。
アイデアざっくり整理してみますね：

コンセプト

* **テーマ**：月食（幻想・一瞬の闇・再び現れる光）

* **雰囲気**：ミステリアスで少しダーク、でもラストは希望の光が差すように

* **ジャンル提案 (Suno)**：

* Electronica / Deep House

* Synthwave / Dream Pop

* Ethereal Disco

サウンドイメージ

* **イントロ**：低くうねるシンセ、遠くで響くコーラス

* **ビート**：ディープハウス系、やや抑えた4つ打ち

* **メロディ**：切ないシンセリード、モジュレーションで揺れる感じ

* **展開**：

1. 月が隠れる → 静かなブレイクダウン

2. 皆既の闇 → ベースとリバーブの深み

3. 再び月が現れる → 明るいシンセでリフトアップ

曲タイトル候補

* "Eclipse Disco"

* "Shadow of the Moon"

* "Lunar Silence"

* 「月影のディスコ」

この方向でまず「曲タイトル+ジャンル」から決めちゃいますか？

ユーザー入力

おねがい

```
# ChatGPT

ではタイトル+ジャンルをまとめますね

---

### [ ] タイトル候補 × ジャンル

1. **"Eclipse Disco"**  
ジャンル : Electronica / Deep House

2. **"Shadow of the Moon"**  
ジャンル : Synthwave / Ethereal Disco

3. **"Lunar Silence"**  
ジャンル : Dream Pop / Electronica

4. **「月影のディスコ」**  
ジャンル : エレクトロニカ・ディスコ（和の響きも混ぜて）

---

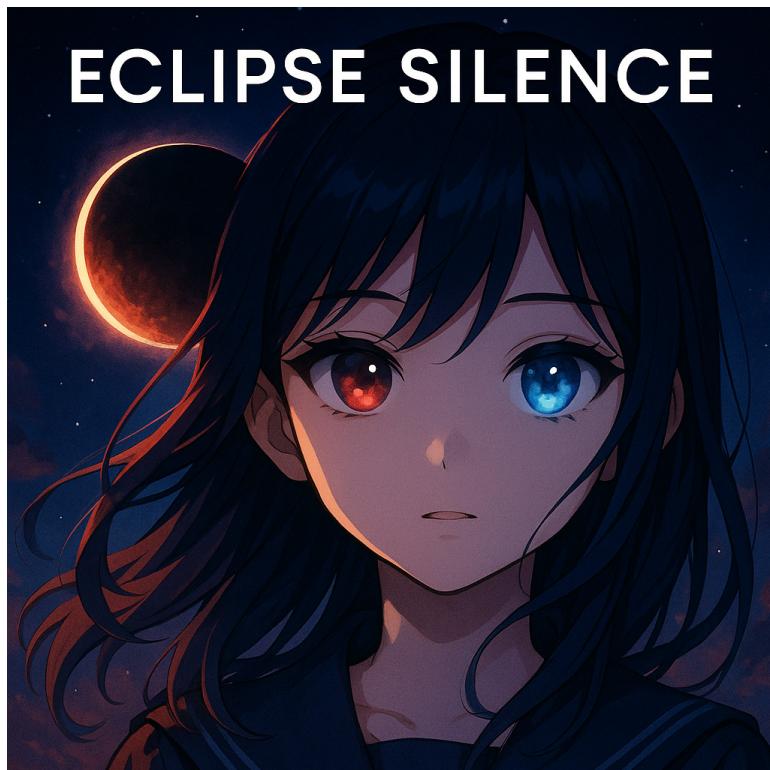
エモくてミステリアスに寄せるなら②か③。  
ディスコっぽさ強めなら①か④がハマりそうです。

どれをベースに作っていきましょうか？
```

・・・この後、やり取りが続きます。

このやり取りで生成した楽曲 :

- Suno : <https://suno.com/song/81d007eb-daff-45e0-ba35-c68c0ec6e7af?sh=IlJJuo0x2TgeuSEm>



▲図 3.6 Eclipse Silence

3.4 おわりに

AIの力を借りていいとはいって、創作活動をすることで、改めてつくづく思うのは、良質なインプットがないと、アウトプットもできないというところです。(そのアウトプットの質は一旦置いといて)・・・人によるとは思うのですが。

これは、イラストや文章などに限らず、プログラムやシステム、アーキテクチャ、またそれらを使った企画・サービスなど、今後AIの力を借りて作られていくことも増えるであろう、IT関連のことにも言えると思います。

エンジニアの皆さん、技術書を始め、ブログやネット記事など、普段から様々なインプットを行っていると思います。

AIが作ったものを正しく評価するためにも、ただエンジニアリングをAIに任せのではなく、AIに、より良いアウトプットをさせるため、AIが作ったものを正しく評価できるようにするために、またその仕組みを作るためにも、人間が、もっと良質なインプットを増やすこと、小さなことでもアウトプットして、外部に知見を広めることなどを行っていくことが、今後さらに重要だと、つくづく感じています。

創作しましょう！！！

きっと生活が豊かになります。

Nov.2025 sakasama

第4章

コード速読術

「コードを読んでるんだけど、なかなか読み進められない・・・」そんなことを思いながら、コードを読んでいませんか？

私も昔はそうでした。修正や障害対応の調査のため、コードを1行1行、場合によっては1文字ずつ読んでいましたが、なかなか終わりません。それに読む場所が収集つかなくなり、「結局、何を調べていたんだっけ？」と迷い、調査が終わらず、徒労感に苛まれるが多々ありました。

私はITエンジニアであるため、IT技術の本を読むことが多いのですが、読む速度も遅く、苦労していたため、通信講座で速読を勉強しました。その結果、ある程度早く本を読めるようになりました。それに伴い、コードも素早く読めるようになりました。

コードが素早く読めるようになると、以下の業務も素早く終わるようになりました。

- コードレビュー
- 技術記事などのキャッチアップ
- コードの調査・解析
- コードのバグの特定

コードの速読は、コーディングの速度を高め、知識を得るためにチートスキルがあります。その思いをこの章で記載していければと思います。コードを素早く、かつ正確に理解するための具体的なテクニックとマインドセットを習得するキッカケになれば幸いです。

4.1 なぜ、あなたのコードリーディングは遅いのか？

4.1.1 「コードを読むこと」と読書は違う

小説を読書する場合、1行ずつで読んでいきます。通常は、コードも同じように1行ずつ丁寧に読むかと思いますが、非効率です。脳は文字の羅列を追うのではなく、意味のまとまりを認識するようにできているからです。またコードはスコープ単位で動作を記載するため、1行ずつ読む必要はありません。以下のコードを例に解説します。

▼リスト 4.5 ファイルを読み、標準出力へファイルの内容を出力

```
1: String filePath = "sample.txt";
2:
3: try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
4:     String line;
5:     while ((line = reader.readLine()) != null) {
6:         System.out.println(line);
7:     }
8: }
```

上記はファイルを読み、標準出力へファイルの内容を1行ごとに出力するコードです。

このコードを1行ごとに切り分けていきましょう。

▼リスト 4.2 ファイルを開く

```
1: try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
```

上記箇所でファイルを開きます。

▼リスト 4.3 ファイルを1行読む

```
1:     while ((line = reader.readLine()) != null)
```

上記箇所でファイルを1行ごとに読みます。

▼リスト 4.4 ファイルを読むサンプル

```
1:     System.out.println(line);
```

上記箇所で読んだファイル1行分のデータを標準出力へ出力します。

この通り、1行ずつ読むと上記のような感じにありますが、とても時間がかかります。「ファイルを読み、標準出力へファイルの内容を1行ごとに出力する」コードですから、1つの固まりとして捉えて、読む方が効率的です。「ちゃんと読まないと、そんなコードなのか、分からぬじやない！」という意見が出ると思いますが、その通りです。w そのご意見については、次のセクションで説明します。

4.1.2 完璧主義が邪魔をする

コードを読む時、全てを完全に理解しようとする必要はありません。コードの多くが定型の記述であり、8割程度の理解で、該当のコードが何をしているか、理解することが可能だからです。そのため、コードを完璧に理解する必要はありません。

▼リスト 4.5 ファイルを読み、標準出力へファイルの内容を出力

```
1: String filePath = "sample.txt";
2:
3: try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
4:     String line;
5:     while ((line = reader.readLine()) != null) {
6:         System.out.println(line);
7:     }
8: }
```

前のセクションで見せたコードです。Javaのご経験が豊富な方はすぐに読めて、どのような処理をするか判断できると思いますが、ご経験や知識が少ない方は難しいと思います。が！ 上記の通り、完璧に理解する必要はありません。「ファイルを読んで、出力しようとしているんだろうな～」という程度の理解で良いのです。

4.1.3 コードの「全体像」が見えていない

「木を一本ずつ見ていては、森全体は見えない」という言葉があります。コードを読む時、特定の処理、特定の関数、特定のクラスだけを着目して読んでいては、コード全体の動作を理解することにとても時間がかかることがあります。そのため、最初にファイル構造や主要なクラスの役割を俯瞰し、だいたいの動作を抑え、頭の中で処理の大まかな処理フローを作成することが重要です。

前のセクションにも通じますが、コード1行1行を見ていては、時間が非常にかか

4.1 なぜ、あなたのコードリーディングは遅いのか？

り、非効率です。そのため、全体を捉えるように、ざっと読んでいくことが望ましいと考えています。もし気になったら、そのときに気になったコードを注視して、丁寧に見ればよいのです。

4.2 コードを「スキャン」するための視点

前章ではコードの速読をするための心構えについて書いてきました。本章では、コードを読み込み、取り込んでいく「スキャン」の視点について書いていきます。

4.2.1 目的別に読む

コードを読む目的が明確だと、読む速度も向上します。体表的な例を挙げて、説明します。

バグを直したい

怪しい部分に絞り込み、ログやテストコードから逆引きすると、早めにバグを特定しやすくなります。

機能を実装したい

類似の機能や関連するインターフェースを中心に探すことも有効的です。

レビューしたい

変更差分とコミットのメッセージのみに集中して読むと、読む範囲が限られてくるため、素早くレビューすることが可能です。

4.2.2 コードを「色」と「形」で捉える

Visual Studio Code や Atom、IntelliJ IDEA など優れた IDE の機能によって、コードの可読性を高めることもコードの速読にとても有効です。以下が有効的な機能です。

- IDE のシンタックスハイライトを最大限に活用する
- 予約語、文字列、変数など、色で情報を区別する
- インデントや空白行によって作られるコードの「形」から、ブロック構造を理解する

4.2.3 名前からコードを読み解く

優れたコードは変数名・関数名・クラス名などに、意図や目的が明確で分かりやすい名前をつけています。そのような変数名・関数名・クラス名を見ると、変数名・関数名・クラス名を見るだけで、「何をしようとしているのか」を語っていることが多

4.2 コードを「スキャン」するための視点

いです。読みやすい名前だと、コードを読み進めるだけで、処理の一連の流れである「ストーリー」が頭に入ってきます。

多くのプロジェクトや現場では、命名規約を定めていますが、上記のように「ストーリー」となるような命名規約が望ましいです。

4.3 コードリーディングを加速させるテクニック

今までではコードを素早く読むためのマインドが中心でしたが、この章ではコードリーディングを早く読むための具体的なテクニックについて書いていきます。

4.3.1 テンプレートを見つけ出す

- おなじみのフレームワークの規約や、デザインパターン（ファクトリー、ストラテジーなど）は、実装の詳細を読まずとも意図を推測できる強力な手がかりとなります。
- 繰り返し現れる定型的なコードパターン（`for...range` や `if err != nil` など）は、見つけた瞬間に「意味のチャンク」として脳にインプットしておくと、素早く読むことが可能です。

4.3.2 コードの文脈から推測する

以下の点を推測しながら、コードを読むと、コードの背後にある「流れ」を想像でき、読解スピードが格段に高まります。

- この関数はどこから呼ばれているのか？
- この変数はどこで値が設定されているのか？

4.3.3 脳の負担を減らす

コードの量が多かったり、難解な処理の場合、とても疲れます。以下のようなテクニックで、脳の負担を減らすことによって、読解スピードを高めることも可能です。

- 複雑なロジックは、メモ帳やホワイトボードに図示してみる
- 一度に多くのコードを読まない
- 「今知りたいこと」だけに焦点を絞る

時間や範囲を決めて、段階的に読むことも有効的です。

4.4 ツールがもたらす超速読の世界

コードは IDE や AI を使うと、より速読のスピードを高めることができます。そのテクニックを以下に記載します。

4.4.1 IDE は「読むための相棒」

- 「Go to Definition」と「Find Usages」で、コードを瞬時にナビゲートする
- 「Call Hierarchy」で関数の呼び出しツリーを視覚化する
- デバッガーを使って、動的にコードの流れを追跡する

4.4.2 AI アシスタントを使いこなす

- 複雑なロジックや正規表現を、AI に要約させたり、解読させる
- 新しい API の使い方を AI に尋ね、素早く概要を掴む

4.5 おわりに：コードを読むことは、最高の学習法である

速読術は単なる技術ではなく、より良い設計、より良いコードを書くための視点を養うプロセスです。他人のコードを読むことで、自分の引き出しを増やすことができます。本章で学んだことを日々の開発に活かし、「コードを読むのが楽しい」と感じられるようになれば幸いです。

著者紹介

第1章 Takami Touka / @Takat46

- 職業：ABAP屋さん
 - SAP一筋もうすぐ30年
 - 担当モジュールは主にFI・CRM
 - コーディング、カスタマイズ、コンサルティングなんでもやります／やれます
- SNS：
 - Instagram : cato_lux
 - X : @takat46

第2章 そいる / @sh_yoshida

- 職業：Scrum Master/Developer
- GitHub : <https://github.com/121jigowatts>
- Service：
 - 雑談お題ルーレット : <https://smalltalk.vercel.app/>

第3章 Masayuki SAKASAI / @____sakasama____

- 職業：Webエンジニア
- GitHub : <https://github.com/sakasa>
- SNS：
 - X : @____sakasama____
 - Instagram : __pickleees
 - Bluesky : @pickleees.bsky.social
- https://linktr.ee/____sakasama____

第4章 ヘー（SASAKURA Hideyuki） / @heisy

- クライミング・ボルダリングを愛する IT エンジニア
 - いろんなところで登ってるので、是非お声がけください！
- らぶ AWS
- GitHub : <https://github.com/heisy3390>
- SNS :
 - X : @heisy

21 時からの勉強会 その 1

2025 年 11 月 16 日 初版 v1.0.0

著 者 21 時からの勉強会
編 集 21 時からの勉強会
発行所 21 時からの勉強会

(C) 2025 21 時からの勉強会

