アルタスファイブ通信 2022 年夏号

「ちょっとだけ変なことをしている会社＝技術変態アルタスファイブ！」です。

いつも、ご愛顧いただきまして、ありがとうございます。今年も技術が好きそうな新人 3 人が加わりまして、プロパー19 名の会社となりました。これまで以上に、力強い開発会社たるべく、技術を磨いて、”憎いねこの野郎”を目指そうと思います。

さて、2022 年の上半期を振り返って、アルタスファイブの仕事っぷりをご紹介させてください。

今号では、普段の仕事の中で、「スピード」と「品質」について、どのようなことを気にかけているか、そのあたりをテーマに記載しています。

目次

01 Angular のコンポーネント設計と開発スピード

開発部：沢辺

とうもろこしがおいしい季節になってきました。甘くておいしいとうもろこしが売りの近所の野菜直売所でもとうもろこしが陳列され始め、連日繁盛しています。

最近はとうもろこしに片栗粉をまぶしてから、油であげて食べるのにはまっている、開発部の沢辺です。

とうもろこしを油であげると、粒が取れやすく歯に挟まりづらいのでおすすめです。

今回は私の担当している Angular プロジェクトで、開発スピードを UP させることについて気を付けていることをお話させていただきます。

Angular はコンポーネント指向のフレームワークなので、開発スピードをあげる上ではコンポーネントの設計が肝となります。

共通で使うような箇所を共通コンポーネントとして再利用することが主ですが、一度作成した共通コンポーネントは、テスト済みなので品質についてもある程度保証されたものとなるのも良い点です。

コンポーネントの設計はできるだけ細かく行うことが推奨されていますが、細かすぎても管理が難しくなってしまうと感じていて、

チーム内で連携が取れていないと同じような共通コンポーネントが作成されてしまうということも多々あるため、ここの加減が難しいところになります。

プロジェクト内である程度の粒度は決めていますが、開発者によってブレもありますので、実際にプロジェクト内でもうまく統一できていません。

今は共通コンポーネントを wiki にまとめたり、コードレビューを通してある程度一定の粒度を保てていますが、もっと改

善できることはあるのではと感じています。

設計については長年やっていれば、感覚である程度できるようになるとは思いますが、システム運用をしていくと後から共通コンポーネントとしたくなることも多々あり、将来的なことを考えての設計は中々に難しいものです。

今後はプロジェクト内でのコンポーネント設計のガイドラインを策定することで、コンポーネントの粒度の統一を今以上に行い、更なる開発スピードの UP を画策しています。

まずはこの内容をチームメンバーに共通し、どのようにすれば設計しやすくなるかを議論することがスタートとなりそうです。

前回、Angular の Version Up についてお話させていただきましたが、前回の時は Ver9 だった Angular も現在では Ver14 まで来ています。

プロジェクト内で使用しているアプリケーションの Version は、IE がターゲットブラウザのため、 Ver11 以降は IE が非推奨になるため Ver10 で止まっていましたが、

IE のサポートが終わったことで今後　IE がターゲットブラウザから外れることとなりそうなので、最新バージョンへの更新が検討できることになりそうです。

私的には IE で Angular などの SPA のサイトを動かすことは、動作が非常に重く Chrome などの他のブラウザと違った動きをしてしまうことも多々ありましたので少しほっとしています。

あとがき：

このプロジェクトは数年携わってますが、長期の案件で効きそうなのが、”テスト”ではないかと思います。これまでにテストにかけたトータル時間を考えると、”テスト”を最適化することで生産性をより高められるのではないか、また同時に品質もあがっていくことが期待できます。

弊社でもテストを科学してもっと効率のよいやり方を模索していきたいですね。（開発部 佐藤）

02 品質と速度と二兎を追う実験と

開発部 小池

ただただ熱い日々が続いていますが、皆様ご体調など崩されておりませんでしょうか？

6 月としては規格外の気温にエアコンが怨嗟の唸りを上げるのを聞いて、いかほどの電気代が請求されるのかと戦々恐々としています

気休めに Amazon のセールで購入した Nature Remo E を使って電力消費を観察してみました。無慈悲な数値に背筋だけは寒くなったようです

そんな中、ちょっと役に立ちそうなもの作りたいと思うところがあり、「OSS でなんか書くか」などと衝動的にキーボードをカタカタ鳴らしていました

というのは半分以上嘘で、アルタスファイブ通信のネタに困って、文章思いつかないならコード書いてやり過ごそうなどと怠惰なことを考えたわけでした

今回のお題は開発速度と品質とのことでしたが、問題解決といったら分割統治だろう！と無理矢理こじつけてライブラリを作ってみます

最近ずっと取り組んでいるのが Go 言語なので、頭を切り替えなくても良い、そして志を低空飛行にして Go のモジュールなどを作ろうと考えました

さて書こう

…

……

……作ることを目的としてしまったので手が動きません

問題解決のためのライブラリですが、直面している仕事に直接関係しない問題ってなかなか思いつかないのでした

「よしまた今度だ！そのうちアイデアが出るだろう」

そして時間も経ち、いっそ忘れたことにしてしまって怒られた方が良いかなーなどとダークサイドのささやきが聞こえ始めた頃、ネストが深いデータをいじるのが面倒くさいと感じることが本業のことでありました

ｺﾚﾀﾞ!!

コンセプトとして XPath 風に書いてフィールド値が操作できるものとしました

ポンチを膨らませてみます

```
// 例えばチーム内のメンバ名を集めたい
// こんなのが
var memberNames []string
for _, m := range(team.Members) {
    memberNames = appenc(memberNames, m.Name)
}

// こんな風に書けたら良いな
memberNames := Get(&team, "Members[*].Name")

// 一括で値を変更できるとか...
Set(&team, "Members[*].Name", func(current string)
string {
    return strings.ToLower(current)
```

```
})

// フィールドの値で絞り込めたり...
admins := Get(&team, `Members(Role="Admin")`)

// Reduce できたり
names := Reduce(&team, `Members(Role="Admin").Name`,
func(a string, b string) string {
    return a + "," + b
}, "") // Alice,Bob,Charlie
```

夢が広がる気がしてきました

4 行くらい書いてたのが 1 行くらいになるようです

地味な気もしますが、細かな積み上げが効率を上げるに違いないと信じて…

まずこのアイデアが実装可能か確認します

リフレクション多用するし、ネストが深い場合に型情報が保持されてるかどうか……ちょっと不安はあります(アルタス通信のネタがなくなっちゃう！)

なので実証コードを書いて確認します

あと実際に使いやすいか使用イメージもここで確認します

(カタカタカタターン!!)

どうやらうまく実装できそうです

go のライブラリは github でそのままリリースできます

リポジトリ名がライブラリ名になるので、他の有名どころのライブラリと被らない名前を見繕います

パスで操作ができるから gopath にしようかな… あ、GOPATH って GO 言語のインストール先ディレクトリを示す環境変数でしたね

この名前では混乱するので、構造体の値を操作するライブラリということで goval としました(適当すぎた気もしてます)

最初はスコープを小さくして Get と Set を実装することにしました

フィールドの絞り込みの実装も大変そうなので、最初の実装には入れ込みません

あとはひたすら動くところまで実装します

関数型っぽく Each を実装して、それを拡張する形に Get と Set を実装して……ユニットテストと並列して実装していきます

当初目論みより面倒くさい問題が発生して頭を抱えたりしましたが、とりあえず目指していた機能を満たしたものが出来上がりました

```
team := Team{
    Name: "TEAM-A",
    Members: []*Member{
        {
            Name: "Alice",
        },
        {
            Name: "Bob",
        },
    },
}
// 値を取得する場合
path, _ := goval.Parse("Name")
fmt.Println(goval.GetAll[string](&team, path)) //
[TEAM-A]
path, _ := goval.Parse("Members[0].Name")
fmt.Println(goval.GetAll[string](&team, path)) //
[Alice]
path, _ := goval.Parse("Members[*].Name")
fmt.Println(goval.GetAll[string](&team, path)) //
[Alice Bob]
// 値を設定する場合
path, _ = goval.Parse("Name")
goval.Set[string](&team, path, "TEAM-B")
fmt.Println(team.Name) // TEAM-B
path, _ = goval.Parse("Members[*].Name")
goval.SetFunc[string](&team, path, func (v string,
pathInfo goval.PathInfo) string{
    return strings.ToLower(v)
})
fmt.Println(team.Members[0].Name) // alice
fmt.Println(team.Members[1].Name) // bob
```

とっとととライブラリを公開することにします

その前に忘れてはいけないコードのライセンスですが、こちらは MIT Lincense としました

Github だとリポジトリの作成時にライセンスファイルを追加してくれるので、一般的なライセンスであれば Github にリポジトリを作成してから、clone して作業を始めると良いです

go の場合、git でコミットにタグを付けることでバージョンをリリースすることができます(バージョンがなくても使うことはできます)

早速タグを打ちます

git tag v0.1.0

git push origin v0.1.0

リリース版というには貧弱だし Interface も変えるかもしれないので、メジャーバージョンはまだ 0 としました

https://github.com/tadjp/goval

業務上では抽象度の高い共通処理と言われるものを書くことがあると思います

ただ、それを勝手に切り出して公開するわけにはいかず、
日々の仕事にも追われ私的なコードを作成することがないエンジニア諸氏も多いのではないでしょうか？

go 言語には公式リポジトリのようなものはなく、github にリポジトリを作成してしまえば、手軽に公開できるため、私のように個人的に書いたコードを公開するのが苦手なタイプの技術者にも心理的な障壁が少ないと思います

さて、弊社には目標達成時にポイントをもらえる制度があり「オープンソースの公開」も目標にあります

そして今回のアルタス通信も目標の一つだったりします

この記事で 2 兎を追うことで「目標達成の品質と速度」を達成できるでしょうか!?

結果はこっそり報告させていただきたいと思います

あとがき：

私も、超ニッチなリポジトリをいくつか公開しています。中には star をもらっているリポジトリもあったりするのだけど、仕事以外でプログラミングすることにストレスのない人は、たくさん OSS を公開した方がよいですね。それくらいド変態な方がよいと思います。（開発部 佐藤）

03 コードレビューにおける品質担保と開発速度の関連について学んだこと

開発部 雲居

連日の茹だるような暑さの中世界水泳を見て興奮はしつつもプールは気持ちよさそうだなあと少しは涼しい気持ちになりながらも、10 代の若い選手の活躍を見て私もまだまだ会社の若手として頑張ろうと思っている開発部の雲居です。

私は今 Android のカメラアプリ開発に携わっております。ちなみに Android の開発経験はまだまだ浅いです。

今関わっている案件ではタスクの実装を進めてローカルでの確認を終え PR を出した後にコードレビューを行っております。コードレビューには色々な意味合いがあると思いますが、今回はタイトルにある通りコードレビューで品質担保と開発速度の関連について述べたいと思います。

品質担保という意味ではコードレビューは重要なフェーズになります。潜在バグの発見や MVVM の思想に沿っているか、テスト漏れの発見などなどあげればキリがないのではと思います。今の案件では Android 開発経験者(熟練者)のかたにレビューを依頼してレビューしてもらっております。(本当は相互レビューができれば一番ですが、私はまだまだで目下勉強中です。)

そんなレビューとレビュー対応を進めていく中でいくつか問題が見えて来ました。それは

レビュアーも同じ案件の開発者のためレビューとレビュー対応に時間が取られてしまうことでレビューがボトルネックになってしまうことが発生いたしました。品質担保の観点では問題ないのですが、開発速度の観点では問題になります。そこでレビューに時間が想定よりかかってしまう要因を確認したところいくつか上がりましたが、大きいところを 2 点あげたいと思います。

・どこまでレビュー指摘するか

・PR のコメントやコードコメントが足りていない

１つ目に関しては案件としてレビューの指標・観点がないことで各々のレビューアー任せになっているためレビューアーの負担が大きくなっておりました。そこで最低限の指標として仕様との齟齬、潜在バグやバグに繋がりそうな箇所、コーディング規約に違反していないかをレビュー観点とする、その他の指摘事項は「Nit:」をつけてコメントいただいて対応を協議するようにするといたしました。(今までも「Nit:」いただいておりましたが、こちらもレビュアー任せになっていた。)その他の指摘事項もコメントしていては開発速度に影響するのではと思われるかもしれませんが、レビュイーにとっては「Nit:」のコメントもありがたく成長に大きくつながるため将来を見越して「Nit:」のコメントの時間は必要かと考えます。

2 つ目に関しては主にレビュイー側の問題になります。PR のコメントは仕様との齟齬確認に影響がでます、コードコメントは保守や可読性に影響してレビュアーの負担に繋がります。こちらに関してはレビュイーへの共有を行い各自で PR 依頼前に確認をすることで対応していくこといたしました。

その他私個人として経験不足に起因する指摘はレビュアーに余計な負担になりますので、再度同様の指摘を受けないように吸収していくことでレビューのボトルネックに影響が出ると思いますので対応をしっかりしていきたいと思いました。

その他 PR のやり取りでは時間を余計に消費すると思われることはレビューアーとレビュイー双方でチャットや打ち合わせで疑問等を解消するようにしてやりとりと時間短縮を行うこ

とも取り組んでおります。

上記対策の効果は今の時点ではまだ比較できておりませんので、また別の機会に記載できればと思います。

レビューによって開発速度を落とさずに品質を担保するためにはユーザに価値を届けつつ、負債を残さないバランスを保つという考えを意識して取り組むことが大事になってくるのではないかと思います。レビューがボトルネックになって開発速度を優先しレビューが疎かになっては本末転倒です。しっかり開発者間で意見を出し合いバランスを見極めていかなくてはいけないと思っております。

今回私が述べたことは案件関係なく今後の開発にも当然活きてくると思います。レビュイーだけではなくレビュアーとしてもどんどん鍛えて引き続き邁進

していきたいなと思った次第です。

これを読んでいただいた方の何か気付きに繋がれば良いなと思いながら今回はここで締めさせていただきます。

あとがき：

昨年、Google のレビューガイドを模して、弊社流レビューガイドを策定しました。社内のプロジェクトに適用し始めていますが、すべてに浸透させるまでには、もう少し時間がかかりそうです。また正しくレビューすること自体が、スキルを求めるので、そこの教育も必要ですね。（開発部 佐藤）

04 Django の開発スピードと品質

開発部 西岡

入社 3 年目の西岡です。コロナが一度落ち着いてきており、仮想通貨全般の価格下落や、イーサリアムの PoW から PoS への承認システムの移行などによるマイニング需要の低下などで、やっとグラフィックボードの価格が希望小売価格に近くなってきました。コロナが流行り始める頃に、RTX3000 シリーズまでの繋ぎとして購入した 2070Super をまさか、2 年以上も使うことになるとは思いませんでした。しかし、円安などの状況により、また先行き不安になっている気がします。

今回は、今年担当していた案件で使用した Django について書いて行きたいと思います。Django は Python 用の Web アプリケーションフレームワークです。データベースのテーブル作成や、カラムの追加等のスキーマ変更などを自動で行ってくれるためほぼ DB に触ることがなく開発が可能です。

この案件で Django を初めて触ったのですが、上記のためストレスなく開発を行うことができました。また、Django は新しい公式のドキュメントについても日本語が用意されているため、不明点などを調べる際にも、困ることがありませんでした。

この Django の生成するマイグレーションファイルは Python ファイルとして生成されます。中身は Django の API を呼び出して、データベースに変更を加える形式になっています。このとき、例えば、新しくテーブルにカラムを追加する際、NOT NULL かつデフォルト値がない場合、新しく追加するカラムの値が存在しないので、エラーが発生します。開発している段階だと、テーブルの構造の変更は度々起こりますが、テーブルがすでに存在している場合は一度データベースを削除するか、マイグレーションファイルを変更しデフォルト値に適当な値を入れるように変更する必要があります。このようなマイグレーションファイルをコミットする場合チームメンバーに共有しないと各々の環境でマイグレーションを適用する際に手間が発生してしまいます。このようなことを避けるため、NOT NULL 制約を開発時は付与せず、後々に付与するか、開発時にテーブルを編集するのを避け、マイグレーション時にテスト用データをメンテナンスするようにし、マイグレーション適用をデータベースを削除後に行い、テストデータのインポートからすぐ構築できるように環境を整えることも必要かもしれません。

今回、初めて Django を使って、SQL を書かずにモデルからデータベースの操作を行う開発を行いました。SQL について考えずに楽に開発はできたのですが、モデルの操作で裏でどのような SQL 文について把握する必要があると感じました。Django はモデルで取得すると、QuerySet という構造になっており、実際に、モデルのデータにアクセスの際にデータベースから取得されます。開発の際に、このあたりの理解が甘く、多重のサブクエリを発行していることに気付かず、性能テストの際に、クエリ実行に時間がかかっていることに気付き修正する必要になってしまいました。今後更にデータベースの理解を深め、今後に活かしていきたいと思います。

あとがき

Django の実績も増えてきました。見た目とか操作性にこだわらない管理画面主体のシステムだったら、Django でいいですね。

それはさておき、イーサリアムの PoS がグラフィックボードの価格に影響を与えていたとは！

皆様、弊社はブロックチェーンに関連した案件に、すごく興味があります。そのような案件がありましたら、ぜひ、お声

がけくださいませ。（開発部 佐藤）

05 開発スピードと品質向上のための取り組み

開発部 小林

初めまして、開発部二年目の小林です。

コロナ禍の 2021 年に入社し、基本的にテレワークで働いています。通勤時間が無いので朝や夜の時間に余裕ができましたし、豪雨や猛暑の中を歩いて出勤する必要が無いのはとてもありがたいと感じています。

ただ、せっかく時間に余裕が生まれたにもかかわらず、その時間を活かしきれていない気がします。何かしら熱中できる趣味が欲しいと思う今日この頃です。

今回は、現在担当しているアプリケーション保守案件で、品質や開発スピードの向上のため取り組んでいることについて紹介いたします。

スクラム方式

担当案件では、2 週間単位のイテレーションでサイクルを回すスクラム方式を採用しています。

以前は、依頼されたすべてのタスクの中から優先度の高い順にタスクに取り組んでいましたが、緊急の修正依頼や新規タスクの見積もりなどが入ると、現在のタスクを中断してそちらに取り組む必要がありました。

現在は 2 週間単位で期間を区切り、期間の開始時にそのイテレーションで取り組むタスクを決めています。ただし、完全にタスクを詰め込むわけではなく、緊急の修正依頼などに備えて数日分のバッファを持たせています。

スクラム方式を採用したことで取り組むべきタスクが明確になったため、一つ一つのタスクに集中して取り組むことができ、結果として品質向上に役立っていると感じています。

また、イテレーション中に新たな不具合が報告されても「緊急性が高い不具合か？」「予定されているタスクを放置してまで対応すべきか？」という基準で着手の可否を決められるようになり、タスクの優先度づけがやりやすくなりました。

詳細な見積もりの実施

担当案件では、「そのタスクに何人日必要か」という見積もりに時間をかけて取り組んでいます。

以前はタスクの内容を軽く確認して、軽微そうであれば 3 人日、複雑そうであれば 5 日、といった具合に大まかな見積もりを行っていました。しかし、軽微そうと思っていたタスクに着手してみると複数のプログラムが複雑に絡んでいて、結果的に工数がオーバーしてしまうということがありました。

現在は編集対象のプログラムの確認、具体的な修正方針の決定などを行ったうえで、修正やデプロイなどの各工程にかかる時間を算出し、それらを合計して見積もりを作成しています。

プログラムの確認を詳細に行い、工程ごとに時間を算出することで、当初の予定より工数がオーバーするということは少なくなりました。

また、見積もりの段階である程度の修正方針まで決定してしまうことで、見積もり担当者以外の作業者が着手する場合であっても、修正がスムーズに進むようになり、開発スピードも上がりました。

タスクシュート

こちらは案件として取り組んでいることではなく、個人的に取り組んでいることとなるのですが、タスクシュートという仕組みを使用して一日に取り組むタスクを管理しています。

タスクシュートというのはタスクリストに近いものなのですが、その日に取り組むタスクを列挙するだけでなく、タスクの所要時間を見積もったり、実際の所要時間を記録したりするという特徴があります。

所要時間を見積もることで、タスクの量が適切かを判断できたり、どれだけの余裕があるか、またはどれだけ余裕が無いかを一目で判別することができるようになりました。

また、実際の所要時間を記録することで、タスクの消化が進んでいるのか遅れているのか、現在の進行状況が今後のタスクにどう影響するかがわかり、タスクの消化がスムーズに進むようになりました。

さらに、見積もりと実際の所要時間とを比較することで、自分の見積もり感覚が実際とどれだけズレているのかがわかり、より正確な見積もりを行うことができるようになりました。自分の作業効率を正確に把握できるというのは、開発スピードを上げることにとても役立つと思っています。

以上が、品質やスピードのために現在取り組んでいる内容となります。今後も継続して取り組んでいき、より良い内容にしていきたいと思います。

あとがき

弊社では、仕事のフレームワーク化を模索しています。常に一定レベルの品質を担保し、仕事時間を最適化する方法の 1 つには、”判断の自動化”があると思っています。どういう状況のときに、どう判断したのか、そしてそれが正しい判断だったのかを振り返ることで、判断基準とルールをガイドブックにすることができます。明確なルールがあると、思いつきではない正しい判断を、迷いや悩むことなく即断できるようになるため、スピードと品質の両方を獲得できます。

保守案件にスクラムを持ち込んだことや、タスクシュートの取り組みは、仕事のフレームワーク化に繋がっていきそうです。ガイドブックにまとめられるといいですね。（開発部 佐藤）

06 自分が介在しなくても持続するシステムの実現への取り組み

開発部 原

開発部の原です。

今年も激動の NBA プレイオフが閉幕。カリー率いるウォリアーズが優勝したものの、テイタム率いるセルティックスがシーズン初期にはあそこまで完成されたチームになるとは思っていなかったので惜しくも優勝を逃した彼らの次シーズンを期待したい！

と思いきや別のチームに目を移すとドラフトが終わるや否や KD が衝撃のトレード要求、カイリーも移籍可能性ありとつい半年前まで BIG3 が揃った優勝候補と言われていたネッツが見る影もない瓦解を見せていて無常を感じます。オフシーズンも話題には事欠かなそうです。

最近 GAS(Google Apps Script)というサービスを触りました。

本来はスプレッドシートなどの Google アプリの自動化を実現するためのサービスのようですが、web アプリを作成して公開する機能があるという所に興味がわきました。

とあるコミュニティで、日々のやり取りが蓄積されていく中でデータが膨大になり、過去どんな問題が話し合われたのか、どういった経緯で今のような文化が形成されたのかを新規のユーザが追いにくいという課題感がありました。

そこでまず過去データを全収集するクローラーを作成、そのデータをコミュニティに配布し、有志がそこから抜粋しキー情報を登録することで簡単に過去の情報を参照出来るように web アプリを作成しました。配布データ自体は MySQL のものですが(サイズがとんでもない)、サイト運営に利用する登録データ自体は無料枠で運用していくためにスプレッドシートを DB 代わりに使用しました。(数少ない GAS っぽい要素)

運用を始めて幾月か経ちましたが週約 150 人~200 人が利用してくれており、不具合報告や要望と戦いつつ日々ログやダッシュボードを見るのが密かな楽しみです。友人には「株価の変動から目を離せないヤツみたいで気持ち悪い」と揶揄されますが放っといてほしいもんです。

運用していく中で気になり始めたのは自動化に関することです。

最初の頃は web アプリを機能させる中で手動で補助をしてやる必要がある部分が多くあったのですが、登録データが増えていく中でめんどくささの感情が膨れ上がっていきました。

その感情を糧にクローラーも新規データの取得はトリガー実行で動くよう移植するなど、その他さまざまな機能を自動化していきました。自然言語を取り扱うため不確定要素との戦いが続きましたがおおむね調整が終わり、今では僕が何らかの理由で管理が出来なくなっても GAS の提供が終了しない限り半永久的に動き続けるものになりました。

また、自動化の考えを一歩進め、自分が病気になるなり死ぬなりしてもコミュニティの他の人が管理を続けていけるような整備もしたいと考えだしました。

既に書いたコードはコミュニティに対して github で公開しており、内容に関する細かい説明もドキュメント化しました。

こうなってしまうとコミュニティの人たちと僕との違いはサイト運営に使っているユーザの登録データを持っているかいないかの違いしかありません。そのデータも一定期間自分がアプリの様子を確認しに行かないことをトリガーにダウンロード可能になるようになっています。

趣味で作ったものですが、自分がいなくなった後も勝手に機能し続けていくかもしれないと思うとなんだか面白いです。

最近は楽天 NBA のサイトから収集したデータを元に各チームの戦術分析をするアプリの構想が少し浮かんでおり、こねくり回すのが楽しいです。

今後も引き続きよろしくお願いいたします。。

あとがき：

業務外でプログラムを作ったりしているド変態がここにもいました。仕事として頼まれてないからプログラム書かないとか、なんか、おこがましいんじゃないか(笑)、だから頼まれていないことも、いろいろやっちゃう。でも、結局、余計なことがリテラシーを高めて、力をつけるんですよね。

弊社では、ブロックチェーンのリテラシーを高めるために、ひとまず、ピアボーナスのようなものを仮想通貨で始めてみようといった話が進みつつあります。（開発部 佐藤）

07 ブリリアントジャーク（タチの悪い凄腕エンジニア）

開発部　佐藤

45 歳を過ぎたくらいから物忘れが多くなって、たぶん、その頃から、集中力も落ちてくるんです。こっちの仕事をしてみたり、あっちの仕事をしてみたり、1 つのことに集中できなくなって、そうすると、仕事の捗りも悪くて、俺ってこんなに仕事できなかったっけ？って自分自身にガッカリするようになります。初老の時期は老いる自分に悩まされるんだけど・・・。でも大丈夫！50 を過ぎるころには、アホになっていく自分に慣れて、あまり気にならなくなるよ。

さて、生産性について、本題に入ります。

去年の夏頃にやっていたプロジェクトで、ある BP さんに参加していただいたんですが、最初のうちは、いろいろと提案してくださる方で、いい人だなぁという印象の方でした。ただ提案してくれる内容が、ちょっとイマイチと言うか、素人っぽいアンチな内容が多くて、ほとんど取り入れならないものだったんです。いや、提案してくれるのは、ありがたいんですけどね。

そうこうしているうちに、開発も進んで、コードレビューを行う段階になって、まぁ、アンチな人なので、いろいろ指摘がついちゃうわけなんですが、自分のアイデアが否定されると、山のように「何故？」の質問攻めをしてくるエゴ強めな人で、なかなか修正してくれなくて、1 ヶ月以上マージできない状況が続いたことがあったんです。イヤハヤ参りましたというか、疲れました。

ブリリアントジャークとは、優秀だけど周りに悪影響を与える人材の事を指します。この BP さんは、凄腕ではなかったけど、ある種のブリリアントジャークと言ってもよいかもしれません。

ただ、BP さんにしてみれば、せっかくの提案も受け入れられず、レビューで否定されまくるし、そんな状況が続くと、やる気が失せるのも当然なのかもです。その頃のチームの雰囲気は、険悪ではないけど、他のメンバーも、BP さんも、お互いに話すのが億劫になっていて、距離を置いた関係性だったように思います。結局、BP さんは、期間延長を希望されることなく去って行きました。

今考えれば、生産性に影響を与える「心理的安全性」が欠如した状態に陥っていたのだと思います。

“難しい人”が 1 人入るだけで、チームの生産性は、30〜40%も落ちてしまうことがあるそうです。

https://logmi.jp/business/articles/325646

この記事によると、心理的安全性を高めるために必要なのは、帰属シグナルだと書かれています。自分がこのチームに求められていて、このチームに存在する意義を感じ入ることが、帰属意識を醸成し、それが生産性に寄与するのだと。

1、2 ヶ月一緒に仕事したくらいでは、なかなか強固な信頼関係とか帰属意識は作れないでしょう。もっと長く時間をかけて育んでいくものなのだと思います。

常駐案件などで、現場毎にチームが変わるような仕事のやり方をしていると、帰属シグナルは滲み出てこないだろうなぁ。やはり、自社に仕事を持ち帰って、できるだけ、いつもの顔ぶれの中で、仕事をするということが、心理的安全性を、そして、生産性を高めていくのだと、改めて思いました。

リモート勤務だと、職場という”場”がないので、いつもの顔ぶれを意識し難くなっています。コミュニケーションの機会として、出勤、飲み会、あるいは、チャットでの雑談など、考えた方がよさそうです。

08 お菓子紹介

管理部　間宮

今期も日頃の感謝の気持ちを込めまして、アルタスファイブから選りすぐりのお菓子をご紹介させていただきます。

リモート勤務が主流になった今、以前のような開催は難しいのですが、やっております。

「アルタスファイブ　おやつグランプリ」！！！！！

各方面を巻き込み、さらには 4 月に入社した新入社員さんた

ちにも早速ご協力いただきました！

※「アルタスファイブ　おやつグランプリ」とは、普段の生活の中にある「アレ美味しかったよ」を持ち寄って、会社で購入して、月一の帰社日に、パートナーさん含めて、みんなで食べてみて、レビューを付けるという、ただそれだけのイベントです。

今回取り上げるのは、素材・品質にこだわった絶品おやつ。

オススメの食べ方とともに、厳選した 2 品をご紹介していきます！

スイーツファクトリー・スリーズ【安納芋トリュフ】

チョコとさつまいも！こんな組み合わせがかつてあったでしょうか…

簡単に説明すると、チョコレートをかけたスイートポテト。で・す・が！この甘さ、食感は経験したことがありません。コーティングに使用されているのはベルギー産のチョコレート。そこに合わさるのは、ねっとりクリーミーな食感と甘味が特徴の安納芋です。(写真で伝わるでしょうか、このしっとり感！)しかも農薬・化学肥料の使用を抑え、本場の種子島で丁寧につくられたお芋とのこと。

う〜ん！こだわりが感じられます。

〜オススメの食べ方は 3 種類〜

①冷たいまんまでどうぞ！パリッとしたチョコレートと、しっとりなお芋が最高です。

②常温でとろ〜り。少し置いておくことでチョコもお芋もとろけます。マリアージュ。

③温めて芋感倍増！レンジで数秒加熱することで、スイートポテトの味・香り・甘味がより濃厚に。

それぞれで全く違った味わいを楽しめますよ〜！（間宮は①で一口→②でゆっくり味わうのが好きです）

マーロウ　【葉山ビスコッティ】

ビーカープリンで有名なマーロウ。今回ご紹介するのは、イタリアの伝統的な焼き菓子であるビスコッティです！

この「葉山ビスコッティ」はマーロウ独自のレシピで焼き上げたもの。生地には 3 種類の小麦粉をブレンドしているというこだわりが。

アーモンドとクルミを贅沢に使用し、ザクザクとした食感が楽しめます。ポイントはほんのり香るオレンジピール。一気にさわやかになり、飽きずに食べられますよ。

もちろんこのままでも美味しいのですが、いかんせん硬いお菓子です。

解決策としては……浸しましょう！！

〜オススメの食べ方〜

①コーヒーやビンサント(甘いワイン)に浸す・・・どちらも本場イタリアではポピュラーな食べ方。しっとりしみ込んで大人な味わいです。

②牛乳に浸す・・・ビスコッティの風味も残しつつ相性抜群！冷たいままでも、ホットミルクでも◎

③アイスクリームに添えて・・・暑いこの時期にはピッタリではないでしょうか！

以上、全方法試したくて、たくさん食べちゃいそうです。

年々厳しさを増す夏ですが、おいしいおやつを食べて、なんとかなんとか乗り越えましょう！

お菓子を探し続ける旅はまだまだ続きます…！

あとがき

リモートワークなので、試食会がなかなかできず、出社する人が多い日に、まとめて試食したんですが、お菓子を、いろいろ食べると、ちょっと気持ちが悪いですね。(笑)

（開発部 佐藤）

次号につづく