スクラム開発プラクティス集

第1.1版

2018年8月15日

プラクティス	コンテキスト	課題	狙い	やり方	効果
Ex)プラクティス名	チーム構成やプロジェクトの環境	抱えている課題	どのような効果を狙った上で実行したのか	プラクティスの実施方法	行った結果どうなったか、狙い通りだったか
ペアプログラミング	* 2年目2人、3年目1人、パートナー1人の開発チーム・7年目のスクラムマスター * スクラムマスター、パートナーは技術に強い * 2年目、3年目は業務知識があるが、開発の経験はあまりない	* 技術力があるメンバーが作成したコードには業務的な考慮が漏れており、手戻りが多くなっていた * 業務知識が豊富なメンバーが作成したコードはレビューで技術面での指摘が上がり、手戻りが多くなっていた * 各メンバーが出してくるコードの品質は、レビューを行うことで高くはなっていたが、最初から品質が高い状況ではなかった * 技術力が高いメンバーは、コードは書けるが、業務知識不足がネックとなり調査などのタスクは行えなかった * 業務知識があるメンバーはコードを書きたいが、調査などのタスクを行っていた	* 技術力の差を埋める * コードレビュー時間の削減 * 業務知識やタスクの属人化の排除	* 朝会で担当するタスクを決めた後、各人がペアプロで行うタスクを決定する * 可能な限り前日と異なるペアになるようにする * 全てのタスクをペアで実施する必要はない(業務難易度・技術難易度の高いもので優先的に行う)	* コード品質の向上 * 技術力・業務知識が属人化せずチームで共有される * チームメンバーが担当できる領域が拡大する
ペア調査	* 2年目2人、3年目1人、パートナー1人の開発チーム・7年目のスクラムマスター * 2年目の1人、3年目には業務知識はあるが、他のメンバーは業務知識量が少ない * 業務などに関わる調査タスクは2年目の1人、3年目が専任で行うことが多い	* 調査の中で特定分野の知識が増えるが、実施メンバーが専任になっており、タスクの属人化が発生していた * 調査で得た知識を使用するタスクについて属人化が発生また知識を持たないメンバーでのレビューになることが多く品質が低下していた * 特定の人しか知らない資料があり、認識齟齬や資料を探す時間が無駄に発生していた	* 業務知識、調査タスクの属人化排除 * チーム内の業務知識向上 * 成果物品質の向上	* 業務知識のあるメンバーとないメンバーで一緒に調査を行う(ペアプロの調査版のようなもの) * 全ての調査をペアで行う必要はない(業務難易度の高い部分や、属人化が進んでいるものを優先的に行う)	* 業務知識差によるタスクの属人化の軽減 * 調査タスクを行えるメンバーが増えたことにより、調査後の後続タスクの遅れが減少 * 成果物品質の向上
スプリントプランニング（第2部）で設計を全員で行う	* 開発チーム4人、スクラムマスター1人 * チームメンバーそれぞれにスプリントで実施するストーリーを割り当て、個々で設計・タスクばらしを行っている * チームメンバー間で業務知識や技術力の偏りが大きい	* ストーリーに関する設計・タスクばらしを各メンバーに割り当てると、割り当てられたメンバーがそのままストーリーの責任者のようになってしまう結果、そのストーリー専属で開発を進めてしまう * ストーリーによってタスク粒度が大きく異なり、タスクが進めづらかった * ストーリーを割り当てられたメンバーとその他のメンバーで認識齟齬が発生し、レビュー時の指摘や手戻りが多く発生していた	* 設計・タスクばらしの段階での認識齟齬をなくす * チームメンバーがどのストーリーのタスクでも実施できるようにする	* ファシリテーターがホワイトボードの前に立ち、設計をホワイトボードに書いていく * 他のメンバーも一緒に設計を行いながら、別の方法や疑問点などを出していく * ホワイトボードに記載した内容は、印刷機能などを使い紙に落としRedmineのWikiに添付できるようにする	* 設計・タスクばらしの段階で認識齟齬がなくなった * チームメンバーが1つのストーリーのタスクだけではなく、複数のストーリーのタスクを実施できるようになりタスクの属人化の排除に繋がった * 設計段階でメンバーに実装方法がわかっていないメンバーがいても即座にフォローすることが出来た
新領域に関する勉強会	* 2年目2人、3年目1人、パートナー1人の開発チーム・7年目のスクラムマスター * メンバー間で技術力に偏りがあった	* メンバー間で技術差があり、今まで行ってきたものと異なる技術的な分野(画面実装とHTTP通信のレベル間の差異など)だと実装イメージが湧いていないメンバーがいた * タスクを1時間以下まで細かく分割するようにしていたが、実装がイメージ出来ていないので見積もりより時間が多くかかる * 調査に多くの時間を費やしてしまっていた	* チームの技術力向上 * タスク時間の見積もりと実績の乖離を縮小させる * 各メンバーによる調査時間の短縮	* 技術的に強いメンバーや専任をおいてサンプルを作成してもらい、それをベースに実装方法の勉強会を実施 * サンプルにはテストの書き方なども含めて良い	* チームの技術力向上 * サンプルを参考に実装が行えるため、調査時間が短縮され見積もりとの乖離が縮小された * 当初は技術的に強い1名がほぼ専属でサンプルを作成していたが、他のメンバーも積極的に作成を行うようになりチーム全体の生産性が向上した
Wikiに知った情報をoutputする	* エラー等ではまった際の事象・対処法のアウトプットは行っていなかった * 時間が経過するとエラーの対処法についての正確な記憶が残っていなかった	* 開発中にエラーが発生した際に、実は以前に他のメンバーがハマっているエラーだった * エラー解消のための調査に時間を要していた	* エラー解消の時間削減 * チーム内のTipsの充実	* エラー等ではまった場合はなるべく早い段階でRedmineのWikiに記載 * 調査に使用した新しい資料等も記載して情報共有	* Web等で調べるよりもピンポイントで情報が記載されているためエラー解消の速度が向上した * チーム内のTipsが風化することを防げた
タイムボックスの明確化	* 2年目2人、3年目1人、パートナー1人の開発チーム・7年目のスクラムマスター * スクラムの有識者はスクラムマスターのみ	* デイリースクラムは15分で終了させることを目標にしていたが、30分たっても当日に行うことが決まらない、課題に関係のないメンバーまで集まって課題の話をしていた * スプリントレトロスペクティブは1.5時間で終了させることを目標にしていたが4時間かけて全てのProblemに対して改善案を出そうとしたが、具体的に行動できるような案が出なかった * イベントごとの時間が長すぎて、タスク消化にかけられる時間が減ってしまう	* タイムボックスを遵守することにより、タスク消化にかける時間を増やす * 各イベントで何を行うべきかをチームメンバーそれぞれが意識することで有意義な議論・活動を行うようにする	* タイムキーパーの導入 * 各イベントのタイムボックスをチームで共有 * 時間を超えたら強制終了	* 各イベントにかかる時間が減ったことにより、タスク消化にかけることの出来る時間が増えた * 議論が長くなってしまう、次の明確なアクションが決まらず時間がすぎる場合は、情報が足りていないという認識をし、必要な情報を集めるというアクションを取るようになった * 日々の仕事を進める上でもタイムボックスを意識するため無駄な時間を減らす意識が見に付いた
チャットツールの導入	* 朝のデイリースクラムで1日に行うタスクを決める * デイリースクラム後の定例的な報告の場は次の日のデイリースクラムになっていた * 情報の共有は主にメールで行っていた	* メンバーが「今、何を行っている」かを手軽に共有できる環境がなかった * 技術情報や参考にしたサイトのURLを手軽に共有できる・業務中に簡単な質問を行う環境がなかった	* メンバーが各々何をやっているかを手軽に把握できるようにする * メールだと挨拶などの定型文が入り、手間がかかるのでより手軽に情報を共有できるようにする	* メンバーがアクセスできる端末上にチャット環境を構築 * 各メンバーのタイムライン用のルームを作成し、自分が行っているタスクやエラーが発生した場合はそのルームに状況を流す	* チーム内のコミュニケーションが活発になった * 状況・情報共有のハードルが下がった * メンバー毎のタイムラインで行っているタスクやハマっているエラーについてつぶやくと状況が把握しやすい
CI結果のチャットへの通知	* ソースコードを開発ブランチにプッシュするとCIが実行される * CI結果はメンバーが時間を見計らってCI結果画面を見ていた	* CIが終了したタイミングがわからないため、終了したかどうかを逐一確認していた * 別のタスクなどを進めていると、結果確認を忘れ、CIがエラーになっていても放置されていた	* CI結果確認の手間を減らす * エラーが発生した場合の検知速度を上げる	* チャットツールとの連携を行い、CI結果をチャットのタイムラインに流すようにする * 成功/失敗の両方をタイムラインに流すと量が多くなってしまうため、エラーのみを通知するようにした	* CI結果を無駄に確認する時間が減った * CIでエラーが発生した場合の対処が早くなった * 自分が担当していないタスクの影響でエラーが発生しても検知できるため、メンバーが声を掛け合ってエラーをなくすようになった

CleanArchitectureの採用	* モバイルアプリ開発(iOS)の新規開発 * アーキテクチャ、設計方針についても未決定	* モバイルアプリのアーキテクチャとしてMVC、MVVM、クリーンアーキテクチャなどがあるがどれを採用すればよいか悩んでいた * アプリが今後大きくなることはわかっていたが、どの程度の規模になるかまでの予測は立っていない	* アプリ規模が大きくなる見込みがあったので、拡張性が高く疎結合でテストしやすいアーキテクチャを採用したい * DDDの要素を取込んだ設計にしたい	* クリーンアーキテクチャの採用 * クリーンアーキテクチャについては下記を参照 * https://qiita.com/koutalou/items/07a4f9cf51a2d13e4cdc * https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html	* 各レイヤの責務が明確で設計しやすい * レイヤが別れているため、どこのレイヤで何を行うべきかなのかの共通認識が持ちやすかった * レイヤと責務が明確に別れているため、開発を続けていく中でどの部分を修正する必要があるかわかりやすい * 業務ロジックがコード内で散らばらず保守性が高い
KPTAの採用	* スプリント毎に振返り(KPT)を実施している * 振返りで決まったTryについては次のスプリントで実施する	* 振返りでKPTを採用していたが、「〜を改善する」等の曖昧な改善が多く、実際に改善が行われない	* スプリント毎の振返りで出た改善を確実に行うようにする	* KPTにプラスしてA(Action)を追加する * Actionには「〜を改善する」などの曖昧なものではなく、「〜を行う」など行動できる内容まで具体化する * Actionは担当(各メンバーやチーム全員など)を決めて実施するようにする	* Actionとして具体性が出たため、改善が確実に実施されるようになった * 具体的なActionを洗い出すのに時間がかかる傾向があった
ワーキングアグリーメント作成	* チーム内のルールのようなものが暗黙知的に存在している	* チームのルールが明文化されていないためメンバー間で認識違いが発生することがあった * チーム内で意見の対立が発生した場合にどのように決定するのかを決定していないため、議論が長くなりがち	* チーム内のルールを決めることで一体感を高め、自己組織化を促す * ルールを明文化することでメンバー間の認識齟齬をなくす * 決定するためのルールを決めることで、意見の対立が発生した場合にスムーズに議論が進むようにする	* チームでルールを決め、その内容を模造紙に書いてチームの見える場所に置く * ルールは定期的に見直しを行い、常に更新していく * 記載する内容はチームで合意したものであればなんでも良い(朝会の進め方、開発で遵守したいことなど)	* ルールが明文化されていることで認識の違いが発生しづらくなった * チームが合意して作成したルールなので、チーム自身が遵守しようと自主的に動く * 決定のルールをワーキングアグリーメントに含めたため、意見の対立が起きた際にスムーズに物事が決まるようになった
朝会で30分単位にタスクをばらす	* スプリント計画ミーティングのタスクばらしでは、数時間〜最大でも1日で完了させられる粒度のタスクにばらしている * カンバン形式のタスク管理ツールを使用	* 夕会の作業報告時に、その日完了できなかったタスクの残作業量がどの程度か把握しづらい	* 日々の完了/未完了タスクの細かい粒度での可視化	* 1日に実施するタスクを、スプリント計画時にばらしたタスクからさらにばらして30分単位の小さなタスクにする（朝会前、メンバー各自で実施） * カンバン形式のタスク管理ツールにタスクを起票し、朝会・夕会で共有する	* 日々の完了/未完了タスクの細かい粒度での可視化 * 朝会で、タスクの具体的な進め方や順序について、チーム内での認識合わせができる
実装タスク以外にも時間を取るものはタスク化する	* 1スプリント = 2週間のサイクルで実施 * 顧客とのミーティング（成果物レビュー含む）は週次で実施	* スプリント計画ミーティングのタスクばらしで、各ストーリーに対して機能の実装完了に必要なタスクのみを挙げており、顧客とのミーティングでデモを実施するための準備作業が漏れていた	* ストーリーの完了時にすぐにデモができる状態になっている	* スプリント計画ミーティングで、各ストーリーのゴールとしてデモで見せる内容・方法も記載し、その実現方法もタスク化する	* デモの準備作業を「ミーティングの準備」タスクではなく、各ストーリーのタスクとして確実に消化できる * 各ストーリーのゴールの明確化
コードレビューの上限パッチサイズの設定	* 7年目1人、4年目1人、新人1人(すべてプロパー)の開発チーム * 開発は全員実施し、新人以外の2人でレビューアを分担	* （過去PJでの経験から、）1機能の実装完成時に機能全体を一度にレビューする等、レビュー対象の規模が大きいと、レビューアの負荷が高く、見落としも発生しやすい	* レビューアの負荷削減 * 無理なくレビュー密度の低下を防止する	* レビュー対象ごとに、以下の最大SLOCを設定し、開発者は最大SLOCに収まるよう、レビューに出すタイミングやスコープを調整する ** Java+SQL+JSP ：600 ** JavaScript+SCSS ：300 ** Documents ：600	* レビューアの負荷が低い * コードに対して小さなフィードバックループを回せるため、大規模な手戻りを防止できる
デイリースクラムへの顧客の参加	* 開発チームと業務有識者(PO、顧客)が遠隔地にいる * 開発チームとPOを含む業務有識者が、物理的に離れている * 開発チーム内には業務有識者が存在しない	* 業務的な不明点が発生した場合、メールやQAでのやりとりになってしまうと初動が遅い * メールやQAでのやりとりでは認識齟齬が発生した場合に認識を合わせるのに手間と時間が多分にかかってしまい、無駄な工数が発生してしまう	* 開発チームで発生した業務的な不明点の即時解消 * PO、業務有識者との認識齟齬の防止、コミュニケーションの活発化	* 日々のデイリースクラムにPO、業務有識者に参加してもらう * 電話を相手先につなぎ、スピーカーでチーム全体に相手の声が聴こえるようにする	* 簡単な質問であれば気軽に聞くことができる環境が整ったため、手間のかかるQAやメールを打つ必要がなくなり開発タスクに使用できる時間が増えた * 話をしながら認識合わせをしているため、以前より齟齬は減った * PO、業務有識者側から出て来る情報の質が上がった(開発チームに意図が伝わりやすいように情報を提供してくれるようになった)
デイリースクラムへの顧客の参加	* 開発チームは顧客先に常駐している * 顧客自身もスクラムに理解が有り、実践している	-	* スクラムチームの活動状況をオープンにすることによる、軌道修正の迅速化 * プロダクトオーナーの意図の共有	* デイリースクラムに、プロダクトオーナーにも参加頂く	* 狙い通り * 認識齟齬は減り、コンテキストの共有も進んだ (メール等で余計な説明をしなくとも、前提が理解されているのでコミュニケーションはスムーズに進んだ) * 一方で、プランニング時にはなかったタスクを依頼されることもあるため、スクラムマスターの力量がより問われる
ホワイトボードの積極的な活用	* 開発チームの過半数は中国出身で、日本語では意図が伝わらないケースがある * 技術的にも新しい概念が多くあり、各メンバーが自身のすべきことを明確に想起できない	* 実施タスクの内容に齟齬が発生しやすい * 逐一、ゴールのイメージやタスクの実施内容をドキュメンテーションするのは時間的コストが高い また、それを顧客からも求められていない	* 効率のよいゴールイメージの共有	* 日々の業務的な相談、技術的トラブルについて、ホワイトボードに逐一記述して説明する * スプリントプランニングについても、どういう機能を作るのか、どういう画面を作るのかを、ホワイトボード上でプロダクトオーナーと認識合わせを実施する	* 狙い通り

個々のタスクの完了条件の明確化	* タスクはRedmineで管理スプリントプランニング後に、分割したタスクをチケット登録する運用	* タスクの完了条件が、個人によってブレる(タスクAが完了したという報告をよくよく確認すると、実際に実施すべき内容が完了してない) * スプリントプランニングにてタスクを分割し、タスクをすべて完了させても、ゴールが達成できない（個々のタスクの完了条件に、互いに齟齬が発生する）状況が発生しがち	* 個々のタスクで実施すべき内容に対する認識をチーム内で一致させる * タスクゴールにブレのない状態で、開発チームがタスクに取り組める状況を作り上げる	* タスクを表現するTODO用のチケットすべてに完了条件を明記する	* 何をすべきかは明確になり、メンバーにとっては進めやすいという評価 * 特にタスク粒度を細かくするほど、個々のタスクの完了条件を考え、人にわかるように記述するのは負荷が高い →チケットベースではなく、付箋ベースの運用に変更し、完了条件の補足はコミュニケーションの中で行い、負荷を下げることにTryする予定
夕会の廃止	* デイリースクラムに顧客、メンバー(最大で16名)全員が参加 * 夕会の目的は、進捗状況、困っている（詰まっている）点の把握 * メンバーそれぞれに今日の実施内容や困ったことを報告してもらっていた	* デイリースクラムの参加負荷が高い	* 参加負荷の軽減 * 開発チームが価値を出すべき作業(検証作業)への集中	* 顧客協議の上、夕会を廃止 * 詰まっている点は日々のコミュニケーション、および、朝会で共有することに変更 * 顧客への状況共有についても、Skypeおよびメールで代替	* 狙い通り * 顧客も同オフィスにいらっしゃるとともに、スクラムチームの席も非常に近いため、夕会のような場を設けなくとも情報共有が成立している
ペアプログラミング	* 技術的知識・経験にメンバー間で大きな差がある 　* Java実務経験のない顧客プロパーも開発チームに参画 　* 自社若手メンバーも、Java知識・実務経験に乏しい	* 経験・知識の浅いメンバーに、実装タスクを割り振れない	* メンバーの知識・経験の向上 * チーム力の底上げ	* 明示的にペアプロタスクを割り当て	* 目に見える成果はまだ体感できていない（本プラクティスを採用してから日が浅い）
見積作業(プランニングポーカー)へのプロダクトオーナーの参加	* プロダクトオーナー自身もレベルの高いエンジニア（実装まで見ることができる）	* このスプリントでどこまでできる、どこからはできない、という納得感の醸成が不十分	* プロダクトオーナーへの、スプリントのスコープに対する納得感の醸成	* プランニングポーカーに、プロダクトオーナーにも参加頂く * どういうタスクやどういう調整が生じ、どういうリスクがあるのかも含め、内容を検討・共有	* スコープについては納得いただけている
スプリントレトロスペクティブにおけるProblem・Tryの起票と投票	* スプリントレトロスペクティブにはKPTを採用	* KPTの意見だしが閾達に行われない(参加意識が希薄になる、という恐れがあった)	* 参加意識の向上 * 自分たちが自分たちの環境を向上させられる、という意識付け	* 事前に付箋を配り、それにProblemやTry書いてもらってから意見表明をしてもらう * 採用するTryについても、個々人の投票にて決定する	* 参加意識は向上 * 誰がどういう思いを持っているかも把握しやすくなった * コミュニケーションの円滑化にも寄与