アジャイルプラクティス　達人プログラマに学ぶ現場開発者の習慣

Venkat Subramaniam and Andy Hunt 著／角谷信太郎・木下史彦 監訳

ISBN 978-4-274-06694-8

1. 非難してもバグは直りません
誰かの後ろ指をさすのではなく、自分のできる解決策に注力しなさい。大事なことは、意味のある成果をあげることです。(14 ページ)

2. 応急処置の誘惑に打ち勝ちなさい
いつでもきれいな状態のコードを見せられるように全力を尽くしなさい。(17 ページ)

3. 批判するならアイデアになさい、人ではなく
誰のアイデアが優れているかを競うのではなく、解決策を導き出せたことに誇りを持ちなさい。(22 ページ)

4. 正しいことをしましょう
誠実に、勇気を出して真実を伝えなさい。時にはそれが難しいこともあるでしょう。だからこそ勇気が必要なのです。(25 ページ)

5. 技術の変化に付いていきましょう
あらゆる分野のエキスパートになる必要はありませんが、業界がどこへ向かっているのかは把握しておきなさい。それを踏まえて自分のキャリアとプロジェクトの計画を立てるのです。(31 ページ)

6. あなた自身とチームのレベルを引き上げましょう
ブラウンバッグミーティングを活用してメンバーの知識とスキルを高め、チームの結束を固めるのです。プロジェクトにプラスとなる技術や手法にチームの関心を向けさせるのです。(33 ページ)

7. 新しきを学び、古きを捨てましょう
新しい技術を学ぶときには、足を引っ張りかねない古い習慣を捨てなさい。自動車は馬車とは別次元のものであって、単に馬のついていない馬車というわけではないのです。
(37 ページ)

8. なぜ？ と問い続けなさい
言われたことを額面どおりに受け取ってはいけません。問題の根本を理解するまで質問し続けるのです。(39 ページ)

9. タスクはため込まずに対処しましょう
あるイベントとイベントとの間に、安定した反復可能な間隔を保つことができれば、繰り返し発生する一般的なタスクに取り組みやすくなります。(42 ページ)

10. 顧客に決断してもらうのです
開発者、マネージャ、業務アナリストは、ビジネスに重大な影響を及ぼす決定を下すべきではありません。事の詳細を、ビジネスの担い手に理解してもらえる言葉で伝えたうえで、顧客に決断してもらうのです。
(48 ページ)

11. 優れた設計は地図です。少しずつ発展させるのです
設計は、正しい方向を示す道しるべではありますが、土地そのものではありません。具体的な道順を事細かに指定すべきものでもありません。設計（または設計者）に囚われてはいけません。(52 ページ)

12. ニーズに裏打ちされた技術を選びなさい
ニーズの特定が先決です。ニーズを踏まえたうえで、採用候補の技術が問題を解決できるかどうかを検討するのです。どんな技術の採用においても、批判的な視点から問いを立て、その問いにきちんと答えられるかを確かめなさい。 (56 ページ)

13. いついかなる時でもプロジェクトをリリース可能な状態にしておくのです
プロジェクトを常に、コンパイルが通り、実行可能で、テストも通っていて、突然のデプロイ要請にも迅速に対応できる状態にしておきなさい。 (59 ページ)

14. はやめの統合、こまめな統合を心がけましょう
コードの統合はリスクの温床です。このリスクを緩和するには、早い段階から統合に取りかかり、定常的に統合を繰り返し続けるのです。 (62 ページ)

15. 最初からアプリケーションのデプロイを自動化しましょう
自動化されたデプロイの仕組みを用意して、さまざまな構成のマシンへアプリケーションをインストールし、アプリケーションの依存関係をテストしなさい。QA はアプリケーションだけでなくデプロイもテスト対象にするのです。 (65 ページ)

16. 見通しのいい開発をしましょう
開発の間ずっとアプリケーションを見せられる状態にして顧客の意見を聞けるようにしなさい。彼ら顧客を巻き込むのです。1〜2 週間に 1 回はデモを見せて、積極的にフィードバックを求めるのです。 (70 ページ)

17. インクリメンタルに開発しなさい
最小限だけれども、きちんと使える機能を備えた製品をリリースしなさい。各インクリメントの開発では、1〜4 週間周期のイテレーションをまわすのです。 (74 ページ)

18. 実作業を基準に見積りなさい
現実的な見積りを出すためには、チームを実際のプロジェクトで、実際の顧客と一緒に作業させるのです。実装する機能とその予算の主導権を、顧客に握らせなさい。 (77 ページ)

19. 自動化されたユニットテストを習慣にしなさい
優れたユニットテストがあれば問題が起きてもすぐに把握できます。しっかりとしたユニットテストを用意していない段階で、設計やコードを変更してはいけません。
(84 ページ)

20. 作る前から使いましょう
テスト駆動開発を設計ツールとして活用なさい。そうすれば、もっと実用的でシンプルな設計を実現できるはずです。 (89 ページ)

21. 違いがあれば結果も変わります
継続的インテグレーションツールを使って、サポート対象のプラットフォームと環境のそれぞれの組み合わせでユニットテストを走らせなさい。問題が発覚するのを待つのではなく、進んで見つけるようにするのです。
(92 ページ)

22. 重要なビジネスロジックのテストを用意しましょう
重要なビジネスロジックのテストは顧客が自分の手で検証できるようにします。それに加えて、これらのテストは通常の自動化されたテストでも一緒に実行させなさい。
(95 ページ)

23. 残作業を計測しなさい
見当違いの測定基準で自分自身やチームを欺くのはやめなさい。計測すべき項目は、残っている作業です。(98 ページ)

24. あらゆる不満に真実が潜んでいるのです
その真実を見つけ出し、本当の問題に対処なさい。(101 ページ)

25. 小賢しいコードではなく、わかりやすいコードを書きなさい
読む人に意図がきちんと伝わるコードを書きなさい。人に読めないコードが利口なコードであるはずがありません。(108 ページ)

26. 伝えるためにコメントはあるのです
よく考えて選んだ意味のある名前を使って、コードを読みやすくしましょう。コメントはコードの意図や制約を示すのに使います。ひどいコードを取り繕うために使ってはいけません。(113 ページ)

27. トレードオフを積極的に考慮しましょう
性能、使い勝手、生産性、コスト、市場にリリースするまでの期間。これらすべてを検討するのです。性能が許容範囲なら、それ以外の要素の改善に注力しなさい。性能がよくなりそうだからとか、このほうがエレガントだからとか、そういった理由で設計を複雑にしてはなりません。(117 ページ)

28. コードを書くときは編集・ビルド・テストのサイクルを短くなさい
いつまでもだらだらとコーディングを続けるのはよくありません。早めに切り上げて、わかりやすく、シンプルで、保守しやすいようにコードを整理するのです。(118 ページ)

29. うまくいく最もシンプルな解法を考えなさい
パターンや原則、何らかの技術を採用するのは、それらを使わざるを得ないときだけです。(121 ページ)

30. クラスは狙いを絞り、コンポーネントは小さく保ちなさい
大きなクラスやコンポーネント、何でも詰め込んだ雑多なクラスを作りたいという衝動を抑えることが大切です。(124 ページ)

31. Tell, Don't Ask ── 求めるな、命じよ
別のオブジェクトやコンポーネントの仕事を引き受けてはいけません。こちらからは命じるのみです。自分の仕事に専念することが大切です。(127 ページ)

32. コードを置き換えてシステムを拡張しなさい
インターフェイスの取り決めを守ったクラスでコードを置き換えることで機能を追加したり、拡張するのです。そしてコードの拡張は、継承よりも委譲のほうが適切であることがほとんどです。(131 ページ)

33. 問題とその解決策を記録しなさい
解決策を記録するまでが問題の修正作業です。記録があれば、後でまた検索して再利用できます。(135 ページ)

34. 警告はエラーと同じです
警告が出ているコードをチェックインするのは、エラーのあるコードやテストに通らなかったコードをチェックインするのと同じくらいまずいことです。チェックインしたコードでビルドツールの警告が出るようなことがあってはいけません。(138 ページ)

35. 問題を切り分けて攻めなさい
問題に取り組む際には、対象となる部分を周囲から分離することが大切です。大規模なアプリケーションでは特にそうです。
(141 ページ)

36. 発生した例外はすべて対処するか、さもなくば伝播させなさい
「とりあえず」であっても、例外を握り潰してはなりません。コードを書くときには、処理は必ず失敗するものだという前提に立ちなさい。 (144 ページ)

37. 役に立つエラーメッセージを提供しなさい
エラーの詳細を簡単に見つけられる方法を提供しなさい。問題が発生したときには、役に立つ詳細情報をできるだけ多く提供しなさい。ただし、詳しすぎてユーザを圧倒してはいけません。 (149 ページ)

38. スタンドアップミーティングをしなさい
スタンドアップミーティングによってチーム共通の認識を持てるようになります。ミーティングは、短く、集中した、真剣なものにしなさい。 (155 ページ)

39. まともな設計は積極的にコードを書くプログラマから生まれます
本物の洞察は、実際にコードを書くことからもたらされます。コーディングしないアーキテクトと一緒に仕事をしないように。システムの実態を知らずにまともに設計なんてできません。 (158 ページ)

40. 共同所有を大切にしなさい
開発者たちの担当箇所を順に変えていくことで、システムのさまざまな範囲の、さまざまなモジュールを経験させなさい。
(160 ページ)

41. メンターになりましょう
自分の知識をほかの人たちと分かち合うのは楽しいことです。与えることで得られるものがあります。ほかの人たちを刺激して、もっと良い結果を出せるようにしなさい。チーム全体の能力を高めるように努めなさい。
(163 ページ)

42. みんなに問題を解決する機会を与えなさい
解決策を与えるのではなく、正しい方向に導くのです。その過程で誰もが何かを学ぶことができます。 (164 ページ)

43. コードの共有には段取りがあります
ほかの人たちが使えるだけの準備の整っていないコードをチェックインしてはなりません。コンパイルできないコードやユニットテストが通らないコードを故意にチェックインするのは、犯罪的な過失といえます。
(167 ページ)

44. あらゆるコードをレビューしなさい
コードレビューは、コードの品質を高め、エラーの発生率を低く抑えるという点で、非常に有益です。適切に行えば、コードレビューは実用的で効果的なものになります。タスクが終わるごとに、別の開発者に頼んでコードをレビューしてもらいなさい。 (171 ページ)

45. みんなに知らせましょう
自分の状況、アイデア、関心のある事柄などを公開しなさい。ほかの人たちから仕事の状況を尋ねられるまで黙っていてはいけません。 (173 ページ)