1章
初心者から達人への道

問題を生み出したときと同じような考え方をしていては、
その問題を解決することはできない。
――アルバート・アインシュタイン

達人になりたくありませんか？　正しい答えが直感でわかるようになりたくはありませんか？　さあ、一緒に達人への旅に踏み出しましょう。この章では、初心者であることの意味、達人であることの意味――そして初心者と達人の間に位置するすべての段階の意味――について考えます。物語はここから始まります。

昔々、二人の研究者（兄と弟）が人工知能の技術を発展させたいと思っていました。二人は、**人間と同じように**技能を学びそして身に付けるソフトウェアを書きたかったのです（もしくは、それが不可能であることを証明したかったのです）。そのためにはまず、人間がいかにして学ぶのかを研究する必要がありました。

二人は技能習得のドレイファスモデル†を開発しましたが、このモデルでは初心者から達人への道のりで通過しなくてはならない段階を五つにまとめました。これからドレイファスモデルについて考察していきますが、この概念を効果的に利用したのは我々が最初ではありません。

1980年代初頭、米国の看護師がドレイファスモデルの説くところに従って自らの取り組みを修正し、看護師という職業をさらに発展させることに成功しました。看護師が当時直面していた問題は、プログラマーを含む技術者が現在直面している問題の多くとよく似ていました。看護師の方々は大いなる前進を遂げましたが、我々プログラマーにはまだなすべきことが残っています。

以下にあげた見解は、看護師にもプログラマーにも当てはまりますが、恐らく他の職業にも当てはまるでしょう。

† 詳しくは『純粋人工知能批判――コンピュータは思考を獲得できるか』[DD86]を参照。

イベント型理論 vs. コンストラクト型理論

観察した現象の説明に使われる理論には「イベント型」の理論と「コンストラクト型」の理論の2種類があります†。

イベント型の理論が対象とするのは何らかの形で測定ができるもので、測定に基づき検証や証明が可能です。その正しさを客観的に判定できるわけです。

これに対してコンストラクト型の理論は実体のない抽象概念であり、「理論の証明」について議論しても意味がありません。この型の理論は「有用性」という観点から評価されますが、正しさの客観的な判定はできません。それは「リンゴ」と「実存主義」をごちゃ混ぜにして扱うようなものです。物体であるリンゴと抽象概念である実存主義を一緒に扱うことに、意味を見いだすことはできません。

たとえば、人間の心について考えてみましょう。医療用の複雑な画像装置など、機械を使って、脳に関するさまざまな事実を証明することができます。しかし、私には、皆さんが心を持っているかどうかを証明することはできません。心は抽象概念であり、そのようなものが物体として存在するわけではなく、概念でしかないのです。しかし心が人間にとって欠かせない存在であることも紛れもない事実です。

ドレイファスモデルはコンストラクト型の理論です。抽象的な概念であり、これから見ていくように非常に有用な概念です。

- 現場で働いている達人が、ほかの人から常に達人として認められているとは限らず、達人全員が相応の報酬をもらっているわけではない。
- 達人の全員が管理職になりたいと思っているわけではない。
- 一般職員（従業員）の能力には非常に大きなばらつきがある。
- 管理者の能力には非常に大きなばらつきがある。
- チームのメンバーには技能レベルに大きな差異があることが多いので、チームを、入れ換え可能で均一的な人的資源の集まりとして扱うことはできない。

各技能レベルの違いは、単にすぐ下のレベル「より優れている」「より賢明である」「より速い」だけではありません。ドレイファスモデルでは、技能レベルに応

† 『*Tools of Critical Thinking: Metathoughts for Psychology*』[Lev97]を参照。

図1-1 コンピュータの達人(特にUnix関連の達人)は魔法使い(ウィザード)と呼ばれる

じて能力や態度、腕前、考え方が、どのように、なぜ変化していくのかを説明しているのです。

ソフトウェア開発過程の改善を試みた取り組みは過去に多数ありましたが、そのほとんどは失敗に終わりました。これはなぜなのでしょうか？ この事実を説明するのに、ドレイファスモデルは非常に有用です。プログラマー個人に対しても、そしてソフトウェア業界全体に対しても、意義深い改善をもたらす手順を示唆してくれているからです。

ドレイファスモデルとはどのようなものなのか、見ていきましょう。

1.1 初心者 vs. 達人

さて、達人の域に達したソフトウェア開発者のことを何と呼ぶかご存じでしょうか？「ウィザード†」です。ウィザードは摩訶不思議な16進の数字やコマンド、「ゾンビプロセス」、`tar -xzvf plugh.tgz` や `sudo gem install --include dependencies rails` といった神秘的な呪文を唱えることによって、持ち上がった

† wizard。もともとは魔法使い、魔術師などの意。

お茶の子さいさい

プロのオルガン奏者の面接を担当したことがあります。オーディションの曲として、シャルル＝マリー・ウィドールの「トッカータ†」という熱狂的とも形容できる作品を選びましたが、素人の私の耳にはオーディションにふさわしい高難度の曲に思えたものです。

ひとりの女性応募者が実にうまく弾きました。両足がペダルの上を飛び交い、手はオルガンの上下段を見えないほどの速さで行き来し、険しい顔つきから、極限まで集中している様子がうかがえ、汗をかいてもいたようです。すばらしい演奏で、私もそれなりに感銘を受けました。

ところがその後、正真正銘の達人がやって来ました。その女性はこの難しい作品を先の演奏者より少し上手に、そして少し速く弾いたのですが、手足をタコのようにすばやく飛ばしながら、我々にほほえみかけ、語りかけるように演奏したのです。

いかにも簡単そうに弾いていた、この女性が採用されました。

問題を解決していきます。

他のユーザーになりすましたり、最高権力者である root（ルート）ユーザーに変身したりもします。しかも、こういったことをいとも簡単にこなしてしまいます。イモリの目を少々、コウモリの羽の粉末をひとつまみ入れて呪文を唱えれば、あっと言う間に仕事は終わり、というわけです！

一種神秘的な雰囲気が漂うものの、達人が何かをするときにいかにも簡単そうにやってのけてしまうというのは、「けっこうよくある話」です（上であげたコンピュータのウィザードの話は、門外漢にはチンプンカンプンな話なので、特に神秘的に思えるでしょう）。

料理の達人のことを考えてみましょう。小麦粉や香辛料、見習いの担当でどんどん山積みになっていく汚れたフライパンに囲まれた達人シェフは、自分でやるのはお茶の子さいさいでも、どう調理すればよいかをキチンと人に伝えられません。「これを少しとあれをひとつまみ。多すぎてはいけない。そしてよく火を通してできあがり」。こんな説明しかできません。

とはいえ、愚鈍な振りをしているわけではなく、「よく火を通してできあがり」

† 出所が気になる方のために申し上げると、『交響曲第五番ヘ短調 Op.42 No.1』です。

の意味するところを自分ではしっかりと理解しています。湿度、肉の仕入れ先、野菜の鮮度といった「パラメータ」に応じて変わってくる、「ちょうどよい」と「やりすぎ」の間の微妙な違いがわかっているのです。

高度の専門知識を明確に表現するのは困難である

達人にとっては、自らの行動を細かなレベルまで説明するのは難しいことが多いのです。達人の動きや反応は実践の繰り返しから生まれたものが多く、意識せずに行動しています。達人の膨大な経験が蓄積されているのは脳の非言語的、前意識的領域†であるため、他者にしてみれば観察が難しく、本人にしてみれば明確な表現が困難です。

達人が何かをするとき、ほかの者にはほとんど魔法のように見えます。不思議な呪文、どこからか降って沸いたように出てくるアイディア、神秘としか思えない能力を使って、ほかの者にとっては何が問題なのかさえわからないうちに問題を解決してしまいます。

もちろん魔法ではありません。外界の認識方法、問題の解決方法、思い描くメンタルモデルなど、ありとあらゆることが達人と凡人では著しく異なっているのです。

これに対して、オフィスで長い1日を過ごしてから帰宅した料理の初心者は、湿度の微妙な変化、その日買ってきたニンジンの手触りといったものには興味さえ持ちません。レシピに入れるサフランの量が何グラムなのかを正確に知りたいのです（これは、サフランの値段が法外に高いからという理由だけではありません）。

初心者は、肉の重さから判断してオーブンのタイマーを設定する正確な時間を知りたいと考えます。学者ぶっているわけでも愚か者なわけでもなく、よりどころとなる明確かつ状況（コンテキスト）に左右されないルールが欲しいのです。達人ならば手足を縛られてしまって何もできないような状態を、逆に初心者が求めているわけです。

初心者と達人は根本的に異なります。世界の見方が異なり、反応の仕方が異なります。どのように異なるのか、細かく見てみましょう。

1.2 ドレイファスモデルの5段階

1970年代、ドレイファス兄弟（ヒューバート・ドレイファスとスチュアート・ドレイファス）は人間が技能を習得し、極める過程に関して独創的な研究を始めました。

† 意識と無意識との中間的な位置にあって、努力すれば意識化できる記憶等が貯蔵されていると考えられる領域。

ドレイファスモデルは技能ごとに適用される

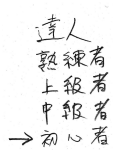
ドレイファス兄弟が観察の対象としたのは、民間航空会社のパイロットや世界的に有名なチェスの名人など、ある分野の技術にきわめて高いレベルの習熟を示した人々です†。兄弟のこうした研究により、初心者から達人へ移行するにつれて、かなりの変化が起きることがわかりました。単なる知識の増加や技能の習得だけではなかったのです。熟練者は、外界を認識する方法や問題への取り組み方、自らが作り上げそして拠り所とするメンタルモデルに根本的な違いを経験するようになります。新しい技能を習得する方法にも変化が現れます。パフォーマンスの向上に貢献したり、その邪魔をしたりする外部的要因も変わってきます。

人物に関するモデルや評価には、人物を「全体として」評価するものが多いのですが、ドレイファスモデルは「技能ごとに」適用するモデルです。つまり状況に依存するモデルであり、個人が元々持つ特性や才能に関するモデルではありません。

人はあらゆることの「達人」であったり、あらゆることの「初心者」であったりすることはなく、むしろ、個々の技能分野について五段階のいずれかに属します。料理人としては初心者でも、スカイダイビングの達人かもしれませんし、その逆もあり得ます。ほとんどの大人は歩行の達人であり、計画を立てることも思索を巡らすこともせずに上手に歩けます。本能となっているのです。大部分の人たちは所得税の申告については初心者です。従うべき明確な規則が十分そろっていれば、なんとか書類を作成できますが、実際何をしているのかはさっぱりわかっていません（そして、税金関連の規則はなぜこうも不可解なのかと疑問に思っています）。

初心者から達人への道には次の五つの段階があります。

第1段階　初心者



初心者は当然のことながら、当該の技能分野における経験を（ほとんど）持っていません。なお、ここで言う「経験」とは、この技能を使うことにより、考え方に変化がもたらされたものを意味しています。たとえ、10年の経験があると主張する開発者であっても、1年の「経験」を単にあと9回繰り返しただけでは、ここでいう意味の「経験」を積み重ねたことにはなりません。

初心者は、うまくやり遂げる能力が自分にあるかどうかをいつも気にしています。指標とすべき経験がほとんどないので、自分の行動のすべてに関してそれがうまく行くかどうか、見当もつきません。学びたいという意欲はそれほどないのが普

† 『ベナー看護論――初心者から達人へ』[Ben01]より引用。

通で、むしろ当面の目標を達成したいという願望を強く持ちます。ミスにどう対処すべきかを知らず、事態が怪しくなると柔軟に対処することが難しくなります。

しかし、状況（コンテキスト）に左右されない、従うべきルールを与えられれば、初心者もある程度効果的に仕事を遂行できます。状況に左右されないルールとはすなわち、「Xが起きたときにはいつもYをしなさい」という形式のルールです。つまり、初心者にはレシピが必要なのです。

初心者にはレシピが必要

コールセンターが機能する理由がここにあります。対象となる事柄について豊富な経験を持たない人々を多数雇い入れ、「決定木（ディシジョンツリー）」の指示に従わせることができるのです。

パソコンを製造販売する大企業なら、たとえば次のようなシナリオを用意することになるでしょう。

1. コンピュータのプラグがコンセントにつながっているかを尋ねる。
2. プラグが入っているなら、コンピュータの電源が入っているか尋ねる。
3. プラグが入っていないなら、プラグを入れてこちらからの指示を待つよう言う。
4. ...

退屈きわまりないものですが、このような明確なルールがあれば初心者はある程度の能力を発揮できます。もちろん、ある状況においてどのルールがもっとも重要かがわからないという問題がありますし、何より想定外のことが起こったときにはパニックになってしまいますが。

多くの人がそうであるように、私も税の申告となると初心者です。ほとんど「経験」がなく、（米国では）毎年個人が税務申告をする必要があるので、かれこれ25年以上同じことを繰り返しているにも関わらず、何も習得していませんし、これに対する考えも変えていません。習得したいとは思っておらず、「今年申告する」という目標さえ達成できたらよいのです。間違いにどう対処すべきかもわかりません。税務署から、簡潔で、どちらかというと尊大な内容の書類が届いても、通常は何が書いてあるのかわかりませんし、どうやって修正するのか見当もつきません†。

もちろん、解決策はあります。状況に左右されないルールが救済にあたります！恐らく、次のようなルールです。

† このような書類は、こうした問題の専門家である（はずの）税理士宛てに、挨拶文と少なからぬ額の小切手を添えて転送します。

図1-2 コーンマフィンのレシピ。でも、正確には何分焼けばいい？

- 昨年の所得を申告書に記入する。
- その書類を税務署に申告する。

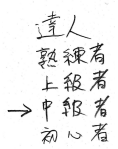
単純明快です。

レシピ（状況に左右されないルール）の問題点は、あらゆることを完全に明示することが絶対に不可能だということです。たとえば、コーンマフィンのレシピには、「20分ほど」焼くと書いてあります。20分より長く焼くのはどのような場合でしょうか。短く焼くのはどのような場合でしょう。焼き上がったとどうしたらわかるのでしょうか。説明のためにルールを増やし、そのルールを説明するためにさらに多くのルールを追加することはできますが、どれほど効果的かつ明確に示せるかには限界があります。いずれは言葉そのものの解釈に行き当たってしまい、昔クリ

ントン大統領が言った「それは“is”という単語が何を意味するかによって変わってきます」というようなことになってしまうのです。この現象は「無限後退」と呼ばれます。ある時点で、明示的に定義することを止めなければなりません。

ルールがあれば始めることはできますが、すぐに限界が来てしまうのです。

第２段階　中級者



初心者のハードルを越えると、「中級者」の観点から問題を見るようになります。中級者はほんの少しだけ、決まったルールから離れられるようになります。独力で仕事にあたることができますが、問題処理にはまだ手こずります。

このレベルの人は情報を素早く入手したがります。たとえばプログラマーなら、新しい言語やライブラリを学んでいるときには、特定のメソッドや関数のパラメータを探すためにドキュメント参照したりする場合があるでしょう。この時点では、細かい理論的な説明を読んだり、基本を最初から学び直したりすることは望んでいません。

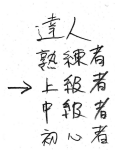
中級者は全体像を見たがらない

中級者は最近経験した似かよった場面を参考にして、コンテキストを考慮したアドバイスの活用ができるようになってきますが、まだ精一杯といった状態です。また、多くに当てはまる原則を見いだせるようになるものの、そこに「全体像」はありません。全体的な理解はしておらず、理解したいともまだ思ってはいません。中級者に全体像を説明しようとすれば、「まだ私には関係ない」などと言って避けて通る姿勢を見せるでしょう。

この種の反応は、たとえば社長が全体会議を召集し、販売予測のグラフや数字を呈示したときなどに目にすることになります。それほど経験のないスタッフの多くは、自分の仕事に関係ないという理由で真剣に考える姿勢を見せないものです。

実際には大いに関係があるわけで、来年も同じ会社で仕事をするかの判断に役立つことなのです。しかし、低い技能レベルに属している人にはそのつながりが見えません。

第3段階 上級者



第3段階になると、問題領域の概念モデルを発展させ、そのモデルを使って効果的に作業ができるようになります。独力で問題に対処できるようになり、それまで直面したことのない新しい問題の解決法を考え出せるようになります。達人にアドバイスを求め、そのアドバイスを応用し、効果的に使えるようになります。

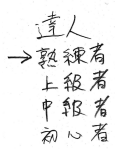
上級者は問題解決ができる

初級者および中級者は物事に対して決まり切った反応をしますが、上級者と呼べるだけの技量を持った人は問題を探し出し、解決します。入念な計画や過去の経験が仕事にさらに生かされるようになります。問題解決の際に、細部のどの部分に焦点を合わせるべきかを決定するにはさらなる経験が必要で、まだ苦労はすることになるでしょう。

上級者の段階に属する人々を形容する言葉としては「指導力がある」「臨機応変な対応が可能」といったものがよく使われます。この種の人々は（公式に、あるいは非公式に）チームの指導者的役割を担う傾向にあります†。チームにとって重要な人となります。初心者に助言を与えてくれますし、過度に達人を困らせるようなことはありません。

ソフトウェア開発の分野では多くの人がこのレベルに達しつつありますが、このレベルに達した人でさえ、理想とされる方法でアジャイルな手法を利用できるようにはなかなかなりません。自らを振り返り、補正するに十分な能力がまだ備わっていないのです。その能力を備えるためには、次の段階である「熟練者」への突破口を開かなければなりません。

第4段階 熟練者



熟練者には全体像が必要です。熟練者は技能を取り巻くさらに大きな概念の枠組みを模索し、理解しようとします。単純化しすぎた情報には苛立ちさえ覚えるのです。

たとえば、熟練者の段階にある人がテクニカルサポートのホットラインに電話して、「電源は入っていますか」と尋ねられたら好意的な反応はしないでしょう（私自身がそのような扱いを受けたら、電話線を通して手を伸ばし、担当者の腑をえぐり出したくなるところです）。

† 『*Teaching and Learning Generic Skills for the Workplace*』[SMLR90]を参照。

達人プログラマーのためのヒント

デビッド・トーマスとともに書いた私の最初の著書『達人プログラマー――システム開発の職人から名匠への道』には、プログラマーという職業にもっとも重要と思われるアドバイスを詰め込みました。

『達人プログラマー』の中で掲げたヒント――この章で言うところの格言――は、何年もかけて蓄積した専門知識を投影したものです。思考力を拡大する慣習を身につけるための「毎年少なくともひとつの言語を学習する」をはじめとして、「割れた窓を放置しておかないこと」「繰り返しを避けること」など、いずれも専門の知識を伝達する上で重要なカギとなるものでした。

熟練者は自己補正が可能

ドレイファスモデルにおいて熟練者は質的に大きな飛躍を遂げます。以前あまりうまくいかなかった自らの行いを修正できるのです。自分の行動を振り返り、次回のパフォーマンスを改善するために自らの取り組みを修正できます。この段階に到達するまでは、この種の自己改善はまったく不可能なのです。

他人の経験からも学ぶことができます。熟練者としてケーススタディを読み、失敗したプロジェクトの噂話を聞き、ほかの人たちの行いを見ることにより、自らが直接の参加者でないにも関わらず、話を聞くだけでそこから効果的に学ぶことできます。

熟練者には、他人から学ぶ能力とともに、「格言」を理解しうまく適用する能力も備わってきます。格言とは、その状況に応じて臨機応変に解釈が可能な基本的な原理・原則です†。格言はレシピとは異なり、特定の状況において特定の解釈がなされて適用されるものです。

たとえば、エクストリームプログラミング（XP）の方法論でよく知られている格言のひとつに、「失敗する可能性のあるものすべてをテストせよ」があります。

初心者にとっては、これはレシピとなってしまいます。初心者は何をテストすればよいのでしょうか？　オブジェクトの属性値を設定する「セッターメソッド」と属性値を取得する「ゲッターメソッド」のすべてをテストすればよいのでしょうか？　プリント文をテストすればよいのでしょうか？　見当違いなものをテストするのが関の山でしょう。

† 『個人的知識――脱批判哲学をめざして』[Pol58]を参照。

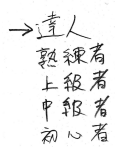
熟練者なら、何が失敗するか――正確にいえば、何が失敗する可能性が高いか――を知っています。経験と判断力を備えているので、この格言が**特定の状況（コンテキスト）において**何を意味するかわかっています。結局のところ、達人になるためには、コンテキストが重要なのです。

熟練者には十分な経験があるので、次に何が起きる可能性が高いかが「経験から」わかります。そしてそのとおりに事が進まないとき、何を変えなければならないかがわかります。どのプランを放棄しなければならないか、その代わりに何をすべきかが、はっきりわかるのです。

同様に、熟練者ならば「ソフトウェアパターン†」も効果的に適用することができます（より低い技能レベルの者には適用できるとは限りません。コラムを参照）。

さあ、いよいよ核心に近付いてきました。熟練者は、アジャイルな手法で中核をなす、ふりかえり（リフレクション）とフィードバックを目一杯活用します。これはこれより前の三つの段階と比較すると大きな飛躍であり、熟練者の段階に属する者は、「上級者よりも上」というよりは、「達人よりもやや下」といった感が強いのです。

第5段階　達人



ついに最終の第5段階、達人にたどり着きました。

どのような分野においても、達人はもっとも重要な知識源かつ情報源です。達人は絶えず、物事を行うよりよい方法を模索しています。達人には膨大な経験があり、それを上手に引き出してピッタリの状況で応用できます。こうした人たちが本をしたため、記事を書き、講演活動を行うのです。達人は現代の魔法使いと言えるでしょう。

統計学的に見て、達人の域に達する人の数はあまり多くなく、恐らく全人口の1%から5%ほどでしょう††。

達人は直感で動く

達人は何かをするのに、「理由があってそうする」のではなく、直感に従って行います。そして、この事実には非常に興味深い疑問が付随することとなります。それは「直感とはなんぞや」という、この本を通して探究していく疑問です。

達人は驚くほど直感的に振る舞います。ほかの者から見れば魔法のようです。いかにして結論に達したのか、曰く言い難いのです。達人には理由がわかりません。「正しいと感じた」だけなのです。

---

† 『オブジェクト指向における再利用のためのデザインパターン』[GHJV95]（通称GoF）で説明されているようなデザインパターン。

†† 『*Standards for Online Communication*』[HS97]を参照。

レシピ

すでに気づいた方もいるかもしれませんが、ソフトウェア開発コミュニティにおける最新の技術は、熟練者あるいは達人の域に達した開発者を対象としています。

アジャイルな開発はフィードバックに依存しており、実際、『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』[SH06]の中で、アジャイルな開発を次のように定義しています――「アジャイルな開発はフィードバックを利用して、共同作業的度合いが非常に高い環境において常時調整を行います」。ところが、以前のパフォーマンスに基づいて自己補正できる能力は、高い技能レベルを持つ者にのみ備わっています。

中級者と上級者はソフトウェアのデザインパターンをレシピと混同することがあり、ともすると悲惨な結果に至ります。たとえば過去のプロジェクトで、前ページの脚注にあげたGoF本に出会ったばかりの開発者がいました。この開発者は興味津々で、GoFデザインパターンを使ってみたかったのです。それも全パターンを同時に、簡単なリポートを生成するだけの小規模なコードの中で。

この開発者は誰かに注意されるまで、どうしようもないコードの中に、GoF本に紹介されている23パターンのうち、およそ17個を詰めこんだのでした。

例として、患者の顔をのぞきこむ医者を想像してください。その医者はひと目見ただけで「この患者はブローゼンプラット症候群だと思うから、この検査をした方がいい」などと言います。看護師が検査すると、医者の診断は実際当たっているのです。どうしてわかったのでしょうか。尋ねてみることはできますが、「何かが変だった」という答えが返ってくるのが関の山かもしれません。

実際、何かが変でした。この医者の膨大な経験、洗練された判断力や記憶力、脳の知的な働きのすべてが、患者の些細な仕草や形態を手がかりにして、診断を下しました。患者の肌の青白さや、体の折り曲げ方かもしれませんが、何が変だったのかは誰にもわかりません。

しかし、達人にはわかります。達人には本質に関係のない部分と重要な部分の区別ができるのです。恐らく意識的なレベルではないのでしょうが、どの部分に焦点を合わせ、どの部分を無視してもかまわないかがわかっています。非常に特殊なパターンのマッチングを瞬時に行ってしまうわけです。

1.3 現場におけるドレイファスモデル

ここまでドレイファスモデルについて細かく見てきましたが、今度はドレイファスの教えを現場に応用する方法を検討してみましょう。少なくともソフトウェア開発の現場においては、このモデルがうまく応用されていない傾向が見て取れるはずです。

達人とて完璧ではありません。達人でも初心者とまったく同じように間違うことはあり、（後ろの章で考察するような）認知的な、あるいはその他の先入観に影響されてしまう場合があることも事実です。また、同じ分野の達人同士であっても、具体的な問題に関して意見が食い違う場合もあるのです。

しかしそれよりひどいのは、ドレイファスモデルを誤解して達人から専門知識を奪い取ってしまうことです。達人を脱線させ、達人の能力を削ぎ落としてしまうのは簡単です。達人にルール遵守を強制すればよいのです。

ルールは達人を破滅させる

ドレイファスの研究においても、これが実証されています。まず、航空会社のベテランパイロットに自らの知識に基づく「ベストプラクティス†」を初心者向けのルールにするよう依頼しました。ベテランたちはルールにまとめ上げることに成功し、初心者はそのルールに基づいて自分たちのパフォーマンスを改善することができました。

ところがこの研究では、達人パイロットをも自らが作ったルールに従わせてみたのです。

その結果どうなったかというと達人たちのパフォーマンスは著しく低下してしまったのです††。

共同作業の場合も同様です。厳しい規則を押しつける開発方法論や社風は、チーム内の達人にどのような影響を与えるでしょうか。達人のパフォーマンスを初心者レベルに引きずり下ろしてしまいます。競合に打ち勝つための「達人の技」がすべて失われてしまうことになるのです。

困ったことに、ソフトウェア産業を全体として見ると、皆がこのやり方で達人を「破滅」させようとしています。「サラブレッドを群れで動かそうとしている」と言ってもよいかもしれません。これは高い投資利益率を上げる方法とはいえませ

† best practice。実践的な行為、行動のうちで、特に効果が高く、効率の向上に貢献するもの。「ビジネス上の（非常に有効な）ノウハウ」のような意味で使われることが多い。

†† 『*The Scope, Limits, and Training Implications of Three Models of Aircraft Pilot Emergency Response Behavior*』[DD79]から引用。

未熟なのに無自覚

熟練していない人の方が、「実は自分はかなりのエキスパートである」と思い込んでしまう傾向があります。

“*Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments*” [KD99] という論文の中で、心理学者のクルーガーとダニングは、白昼に銀行強盗を働いた「自称泥棒」の不運な話をしています。この泥棒はレモン汁を顔につけていれば監視カメラに写らないと思い込んでいたので、自分がすぐに逮捕されてしまった事実を容易に受け入れられませんでした。

「レモン汁男」は自分の仮説の「怪しさを怪しむ」ことをしませんでした。このように正確な自己評価が欠如していること、つまり未熟でありながらそれに気づいていないことを「二次無能力」と呼びます。

この状態はソフトウェア開発で非常に大きな問題となります。プログラマーや管理者に、よりよい手法や方法が存在することを知らない人がたくさんいるのです。私は成功裏に終えることができたプロジェクトに一度も関わったことのない、経験1年から5年程度の若いプログラマーを大勢知っていますが、こうしたプログラマーは「プロジェクトは苦痛なもので、失敗するのが当たり前だ」という考えに初めから囚(とら)われてしまっています。

チャールズ・ダーウィンは「熟知よりも無知の方が自信の源になる」と語っています。

逆もまた真なのかもしれません。本当の達人になれば、いかに自分の持つ知識が乏しいかを思い知らされます。

ん。サラブレッドには他の馬を抜き去ってもらわなければならないのです。

分野を問わず、直感は達人にとってなくてはならない「道具」ですが、「非科学的である」「再現不可能である」と誤解している組織が、直感を軽視する傾向にあります。高いお金を払って雇っている達人に耳を貸さずに、大切なものを捨ててしまっているのです。

これとは逆に、初心者をつかまえて、開発という名のプールの一番深い場所、初心者の足がまったく立たないようなところに投げ入れてしまう傾向もあります。この場合は、羊にサラブレッドの役割を期待していると言えるかもしれません。繰り返します

遵法闘争

本格的なストライキが許されていない産業や状況では、代わりに「遵法闘争」が抗議の手段として用いられます。就業規則で求められているそのとおりの仕事を、それ以上でもそれ以下でもなく行うことであり、一字一句まで従うのです。

その結果、大きな遅れと混乱が起こり、効果的な抗議方法となるわけです。現実世界で専門知識を持っている人は一字一句まで規則に従ったりはしません。もし従えばその作業は確実に非能率的になります。

ベナーは『ベナー看護論――初心者から達人へ』[Ben01]で次のように述べています――「実践（の作業）を完全に客観化、あるいは形式化することは不可能である。なぜなら、実践はリアルタイムに変化する特定のコンテキストにおいて、常に新たになされるものだからである」。

が、これは初心者の効果的な配置ではありません。初心者は（羊のように）「番をしてもらう」必要があり、曖昧さがない明確な指示を与え、小さな成功体験を積ませてあげる必要があります。アジャイルな開発は非常に効果的な開発手法ですが、初心者と中級者だけで構成されたチームではうまく機能しません。

ところが業界のトレンドは、開発者に二つの方向から圧力をかけてきます。すべての開発者を差別しないよう、その能力に関わらず同じように扱うことを強いてくるのです。これにより、達人だけでなく初心者も被害を受けることになります。研究によって値は異なりますが、開発者の生産性には人により20倍ないしは40倍もの違いがあると報告されています†。この現実が無視されてしまっています。

ヒント2
初心者はルールに、達人は直感に従うこと

初心者と達人の違いは、もちろんルールと直感だけではありません。技能レベルの階段を登るにつれて、多くの特性が変化します。この途中で起こる変化の中で

† 1968年、『*Exploratory Experimental Studies Comparing Online and Offline*』[Sac68]において、プログラマーの生産性は、個人により最大で10倍もの開きがあるとの指摘がなされました。どうやら、その差はもっと開いているようです。

も、特に重要な三つを次にあげましょう†。

- よりどころとする対象がルールから直感に移行する。
- 認識が変化し、問題を一様な関連性を有する小部分の集合として認識するのではなく、ひとつの完全かつ無類の統一体として認識し、特定の小部分にのみ力点を置くようになる。
- 問題から遊離して存在する観察者から、システム自身の一部としてそれに関与する存在へと変化する。

これが初心者から達人への行程です。コンテキストから切り離された絶対的なルールから徐々に距離をおきはじめ、直感に重きを置く方向へと向かい、最終的には「システム思考（システムズシンキング）」が説くように、自らがシステム自体の一部分となるのです（図 1-3）。

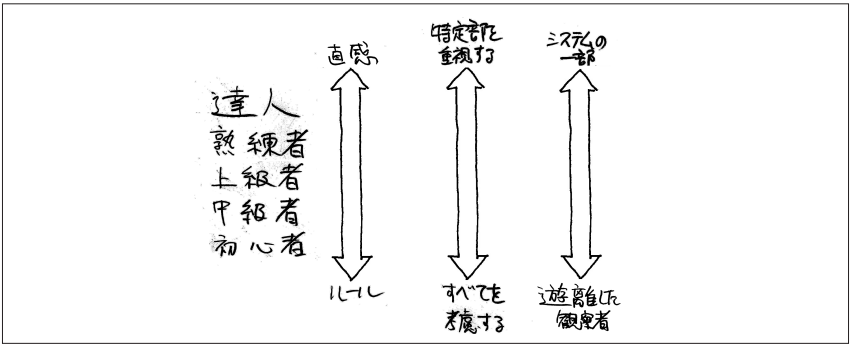


図1-3　ドレイファスモデルおける技能習得

技能分布の悲しい現実

ところで、各レベルに属する人の分布はどうなっているのでしょうか？　皆さんは、この行程のちょうど真ん中に多くの人が位置している、つまり、ドレイファスモデルに基づく技能の分布は正規的であり、典型的な釣鐘曲線を描くと思っているのではないでしょうか。

実際はそうはなりません。

† 『ベナー看護論――初心者から達人へ』[Ben01]で指摘されています。このすばらしい本についてはこれからも何度か言及します。

ほとんどの人は中級者

複数の研究によると、悲しいことにほとんどの人がほとんどの技能について、人生の大半おいて、第2段階の中級者より上の段階に行くことはなく、「必要な仕事を行い、必要になると新しい仕事を学ぶが、仕事を広範かつ概念的に理解することは決してない[†]」のです。実際の分布は図1-4のようになります。

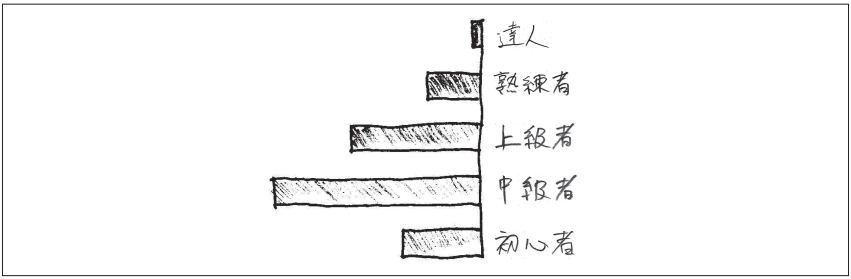


図1-4　技能の分布

（Googleを開発環境として利用したような）「コピー・ペースト・コーディング」の増加や、ソフトウェアのデザインパターンの誤用の蔓延など、この現象の証拠となる逸話は巷に溢れています。

同様に、メタ認知力、つまり自分自身を認識する能力が働くようになるのは、より高度な技能レベルを持つ者に限られる傾向にあります。残念ながら、このことは、低い技能レベルに属する者には自分の能力を過大評価する傾向が見られるということを意味します。最高で50%も過大に評価する人がいることがわかりました。“*Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments*”[KD99]の調査によると、より正確な自己評価への唯一の道は、個人の技能を向上させることであり、これにより結果的にメタ認知力が向上することになります。

自分がどれほど無知であるかがわかっていないことを二次無能力と呼ぶことがあります。初心者は状況が怪しくなってきても自信を失いませんが、達人はこのような状況では、はるかに用心深くなります。自分を疑ってかかる度合いは、達人の方がはるかに強いのです。

† 『*Standards for Online Communication*』[HS97]を参照。

達人 ≠ 先生

達人が常に最良の先生であるとは限りません。指導はそれ自体が専門技能であり、ある事柄に関して達人であるからといって、それを他人にうまく教えられるという保証はありません。

達人は往々にして自ら結論に達した道筋を理路整然と表現できません。この現象を考慮すると、達人よりもやや下の上級者レベルの方が、初心者の教育係としては向いているかもしれません。チーム内でペアリングや指導を行う際には、指導される側に近いレベルの教育係を割り当てるのも悪い考えではないでしょう。

ヒント3
自分が何を知らないかを自覚すること

残念ながら、常に達人より中級者の方が数多く存在します。しかし、底辺に属する人の数が多いとしても、「分布」であることは事実です。運良く自分のチームに達人がいるのなら、達人に便宜をはかる必要があります。同じように、少数の初心者、多数の中級者、少数でも強力な熟練者にもよりよい環境を整えてあげる必要があります。

直感とパターンマッチングが明示的な知識に取って代わる

達人の特徴は直感を使うこと、そしてコンテキストに応じたパターン認識を行うことです。初心者には直感力が欠如している、達人よりも下のレベルの人はパターンの認識がまったくできない、と言っているのではありません。達人の直感とパターン認識は、明示的な知識にとって代わるものなのです。

初心者の「コンテキストに依存しない画一的なルール」から、達人の「コンテキストに応じた臨機応変な直感」への移行は、ドレイファスモデルの核をなす部分です。したがって、この本の残りの部分では、直感をいかにうまく利用するか、そしてパターンをいかに認識し、それをどう利用するかに、多くのページを割くことにします†。

† ここで言う「パターン」は、ソフトウェアの「デザインパターン」のことではなく、一般的な意味で使われる「パターン」を指します。

専門技能の習得に必要な年数は10年？

達人になりたければ、対象分野に関わらず最低10年は努力する覚悟がなければなりません。チェス、絵画、作曲、ピアノ演奏、水泳、テニスといった分野を対象に行われた研究†によれば、モーツァルトからビートルズまで、ほとんどのケースで、世界でもトップクラスと認められるようになるまで、最低でも10年間の精進が必要なことが明らかにされています。

たとえばビートルズが『エド・サリバン・ショー』への記念すべき出演を果たし、世界に旋風を巻き起こしたのは1964年のことです。ビートルズ初の大ヒットアルバム『サージェント・ペパーズ・ロンリー・ハーツ・クラブ・バンド』は1967年にリリースされました。しかし、ビートルズは、1964年のツアーを前にして魔法のように結成されたわけではなく、すでに1957年からクラブで演奏をしていました。『サージェント・ペパーズ』の10年も前のことです。

そして、生半可ではない「本気の努力」が必要です。10年間、ただ取り組むだけでは不十分で、激しい訓練が必要なのです。著名な認知科学者K・アンダーズ・エリクソン博士††によると、このような訓練には次の四つの条件が必要です。

- 明確に定義されたタスクが必要である。
- タスクには適度な難度（やりがい）が必要だが、実行可能でなければならない。
- 周囲から、行動のよりどころとなるような有益なフィードバックが提供されなければならない。
- 復習のための繰り返しの機会が与えられ、またエラーを修正する機会も提供されるべきである。

こうした条件を満たした訓練を10年間着実に続ければ、達人になれるでしょう。『達人プログラマー――システム開発の職人から名匠への道』[HT00]に書いたように、チョーサーでさえ「人生はとても短いのに、技術の習得にはとても長い時間がかかる」とこぼしているのです。

ここで、朗報があります。ひとつの分野で達人になれば、別の分野における専門技能の習得がずっと楽になります。少なくとも「技能を習得するための技術」や「モデルを構築する能力」はすでに身につけていることになるのです。

† 『*The Complete Problem Solver*』[Hay81]ならびに『*Developing Talent in Young People*』[BS85]を参照。

†† エリクソン博士の研究を紹介してくれたジューン・キム氏に感謝します。

1.4　ドレイファスモデルの効果的な利用

1970年代の終わり頃、看護師たちは八方ふさがりの状態でした。当時の問題点を簡潔にまとめると次のようになります†。

- 看護師自身が単なる役立つ**もの**として軽視される傾向がありました。高度に訓練された医師からの命令を遂行するだけで、患者の看護に関する情報や意見は何ら期待されていませんでした。
- 給与体系が不公平なため、達人看護師は患者の直接的なケアを初心者や中級者に委ねようとしていました。管理や指導、講演活動を通じて得られる報酬の方が多かったのです。
- 看護教育が揺らぎ始めました。多くの人が、プラクティスを形式化したモデルが最良の教育方法と考えていましたが、形式的な手段と道具に依存しすぎたために、実践的な現実の経験が減少してしまっていました。
- ついには、真の目標である患者の医療・看護面での成果を見失ってしまいました。どのような方法論に従ったか、その患者を誰が担当したかは別にして、結果はどうだったのでしょうか。患者は生存し、元気に過ごしたのでしょうか。あるいは不幸な結果に終わってしまったのでしょうか。

上の各項目を注意深く読めば、気味の悪いほど心当たりがある人もいるのではないでしょうか。少々手を加えて、ソフトウェア開発の現場を反映した箇条書きに変えてみましょう。

- プログラマー自身がただの役立つ**もの**として軽視されることが多々ありました。高度に訓練されたアナリストからの命令を遂行するだけで、プロジェクトの設計やアーキテクチャに関する情報を提供したり意見を述べたりすることは何ら期待されていませんでした。
- 給与体系が不公平なため、達人プログラマーは実地のプログラミングを初心者や中級者に委ねようとしていました。管理や指導、講演活動を通じて得られる報酬の方が多かったのです。
- ソフトウェア工学の教育が揺らぎ始めました。多くの人々が、実践を形式化したモデルが最良の教育方法と考えていました。形式的な方式と道具に依存しすぎたために、実践における現実の経験が減少してしまいました。

† 『ベナー看護論――初心者から達人へ』[Ben01]で説明されています。いくつかのケーススタディと体験談から導き出されたものです。

- ついには、真の目標であるプロジェクトの成果を見失ってしまいました。どのような方法論に従ったか、そのプロジェクトに誰が関係したかはさておき、結果はどうだったのでしょうか。プロジェクトは成功し、さらに発展させることができたのでしょうか。

こうして変更してみたら、もっとピンとくるようになったでしょう。ここにあげた事柄は実際にソフトウェア業界が現在直面している深刻な問題なのです。

1980年代初頭、看護師がドレイファスモデルの教えを自分たちの業界に適用し始め、驚くほどの成果を上げました。ベナーが自身の画期的な本の中でドレイファスモデルを公開、説明したことにより、あらゆる関係者が自分自身、そして同僚の技能と役割についての理解を深めていきました。この本は、看護師という職業全体について、そのレベルを向上させる具体的な指針を提供していたのです。

その後25年ほどの歳月をかけて、ベナーやこれに続く著者や研究者が看護師という職業を望ましい方向に変えていきました。

ですからR&Dの精神――とはいってもResearch & Development（研究開発）ではなくRip off and Duplicate（盗用して複製する）という精神のことですが――を大いに発揮して、我々は看護師の方々の成果から多くの教えを借用し、ソフトウェア開発（あるいは自分の関与する分野）に適用していけばよいのです。看護師がどのように実行したか、そして我々は何ができるのかを詳しく見てみましょう。

責任の受け入れ

25年前、看護師は何の異議も差しはさまずに、熱意と誇りさえ持って命令に従うよう期待されていました。患者のニーズや状態に明らかな変化があっても無視し、「医師の命令から決して逸脱しない」ことが求められました。

このような姿勢に至らしめた一因は、患者の状態の継続的かつ軽微な変化を観察する立場にはなかった医師にありました。さらに看護師自身も原因を作っており、実務の過程で意思決定の責任を自ら放棄し、権限のある医師に委ねてしまっていました。その方が看護師にとっては職務上「安全」でしたし、また、そうした立場をとることには心理学的な根拠があるのです。

ひとりの研究者が医師になりすまして病院に電話し、所定の患者に特定の薬剤を与えるよう命令するという実験が行われました[†]。その際、病院スタッフの不信感を増すように以下のような細工がなされました。

† 『*Influence: Science and Practice*』[Cia01]で説明されています。

- 処方せんは書面ではなく、電話で伝える。
- その薬剤は、その病院の通常の承認リストに掲載されていないものである。
- その薬剤のラベルの記載に従うと、処方された投薬量は最大量の 2 倍である。
- 電話した「医師」は、看護師やスタッフの知らない人間である。

ところが、こうした明白な「危険信号」にも関わらず、95% の看護師がだまされ、躊躇なく薬剤棚へ足を運び、当該患者に投薬しようと患者の部屋へ向かったのです。

幸い、ニセ医師の共犯者が看護師を呼び止めて実験を説明し、ニセの命令の実行を阻止しました†。

看護師とまったく同じような問題が、プログラマーやアーキテクト、プロジェクトマネージャーなどにも見られます。これまでの慣例では、プログラマーからアーキテクトやマネージャーに対してはまったくフィードバックされないか、されたとしても取り付く島もなく拒絶されるか、プロジェクトの喧噪の中でいつの日か失われてしまうかのいずれかの道をたどります。前述の実験において看護師がとった行動と同様、プログラマーは往々にして眼前の危険信号を無視し、間違いとわかっているものを実装することがあるのです。アジャイルな手法はチームメンバー全員からのフィードバックを奨励し、それを効果的に活用しますが、それでは仕事の半分しか終わりません。

「命令に従っていただけ！」は通用しない

新しい考え方が浸透するに従って、個々の看護師が責任を取ることが必要になりました。同じように、個々のプログラマーも責任を取らなければなりません。ニュルンベルク裁判で見られた「私はただ命令に従っていただけ」という自己弁護は第二次世界大戦でも、看護師という職業でも通用しませんでした。ソフトウェア開発でも通用しないのです。

しかし、こうした変化を成し遂げるには、レベルを引き上げる必要があります。初心者の次のレベルである中級者が独力でこの種の決定を下すことはできません。現在いる中級者に手を貸し、3 番目のレベルである上級者の技能レベルに引き上げる手助けをしなければなりません。

技能レベルの引き上げに役立つ一番の方法は、中級者の周囲に優れたお手本を配置することです。人は自然と別の人のまねをします（6 章を参照）。具体例を見る

† これは昔行われた実験です。今の時代に病院に電話をかけ、ニセの命令を与えることのないようにしてください。そんなことをしたら警察が来ます。

経験の伴わない専門技能は存在しない

ジャズは実世界の経験が大きな意味を持つ芸術です。ジャズの演奏に必要なコードとテクニックを全部頭で覚えても、ジャズの「フィーリング」をつかむには繰り返し演奏するしかないのです。有名なトランペット奏者兼ボーカリストのルイ・アームストロング（サッチモ）はジャズについてこう語っています――「おいおい、人に訊いてばかりいちゃ、いつまでたってもわかるようにはならないぜ」。

経験を伴わない専門技能は存在せず、経験に取って代わるものはありません。しかし、すでにある経験を効率的かつ効果的に活用することは可能です。

ことで学んでいくのです。子供がいる人なら、親の「言うこと」はめったに聞かない子供でも、親の「やること」はいつもまねをしていることにお気づきではありませんか？

ヒント４
観察し、まねることによって習得する

トランペット奏者のクラーク・テリーは、音楽習得の秘けつは次の３段階を経験することであると弟子たちに話していました。

- 模倣
- 同化
- 革新

すなわち、まずは既存の慣行を模倣し、次に長い時間をかけてゆっくりと言葉では表せない知識と経験を自らの体に取り込み同化させます。最終的には、模倣を越えて革新する立場になるのです。これは武道で稽古を積む過程「守（しゅ）・破（は）・離（り）」にそっくりです。

「守」の段階では、門弟は師匠に教えられた技術をそのまま模倣します。「破」の段階になると、意味と目的を熟考し、さらに深い理解に到達しなくてはなりません。「離」は超越を意味し、もう門弟ではなくなった実践者は、模倣ではないオリジナルの考えを提案します。

したがって、熟練者や達人レベルの人が同じプロジェクトにすでにいるならば、何をおいてもプロジェクト内に囲い込んでおく必要があります。熟達者がその分野にとどまらなければ、革新の段階まで到達しようがないのですから。

専門技能を現場にとどめる

25 年前、専門技能を有する大勢の看護師たちが職場を離れる動きが加速しました。給与体系とキャリア開発に限界があったため、高い技能を持つ看護師が一定のキャリアに達すると、直接的な臨床業務を離れ、管理もしくは教育分野への転向や、看護分野からの完全撤退を強いられたのでした。

同じようなことがソフトウェア業界でも頻繁に起こっています。プログラマー（別名「コーダー」）の報酬はたかがしれており、営業担当やコンサルタント、上級管理者などは、チームで腕利きのプログラマーの倍以上の報酬を得ていることも多いのです。

企業は、こうした花形開発者が組織にもたらす価値を、より徹底的に、より細かく調べる必要があります。

勝者は敗者を抱きかかえて前には進まない

開発プロジェクトのメンバーに対して、「チームワーク」とか「共通の目標」といったものの大切さを説くのに、スポーツが比喩としてよく使われます。しかし、実際には、プロスポーツのチームにおける現実は、理想的なチームワークの例とはマッチしない部分が多々あります。

あるプロ野球チームに二人の投手がいたとしましょう。ひとりの年俸は 2 億円で、もうひとりの年俸はわずか 600 万円だとします。このとき問題になるのはこの二人の守備位置ではなく、またプロ野球選手としての経験でもありません。何より、各人がチームにもたらす価値が問題なのです。

ジェフリー・コルヴィンはある記事†の中で、これをさらに詳しく説明しています。現実のチームにはスターが存在します。チームメンバー全員がスターではなく、新人（ドレイファスモデルで言うところの初心者あるいは中級者レベル）もいれば、上級者レベルの選手もいます。優秀な新人は階段を駆け上がっていきますが、このとき勝者が敗者を抱きかかえて一緒に階段を上がってやることはありません。敗者は単にチームから解雇されるのです。最後にコルヴィンは、こうも言っています。「上位 2% に入ったとしても世界一流ではない。本当の世界一流とは上位 0.2% に入る人だ」。

† 『*Fortune Magazine*』 2002 年 3 月 18 日号, p. 50。

そしてこれは競争の激しいスポーツチームに限ったことではありません。（人道主義の本家とも言える）教会においてさえ才能の差を認めて、効果的に利用しようとしています。最近私はある教会の会報を見せてもらう機会があったのですが、そこには音楽関連のプログラムを維持し、発展させる方法についての助言がなされていました。そのアドバイスを以下に記します。

- グループ全体の強さは、もっとも弱い部分の強さと等しくなります。主要サービスにはもっとも上手な演奏者と歌手を集め、その他のサービス向けとして「二軍」を結成しましょう。
- メンバーを変えずに、毎週、固定したグループにしましょう。グループには一体感が欲しいのです。メンバーを頻繁に入れ替えるのは効果的ではありません。
- タイミングが非常に重要です。（バンドにおける）ドラマーや（合唱における）伴奏者は「本物」でなければなりません。あてにならないドラマーやオルガン奏者を使うくらいならば、録音を使った方がましです。
- 才能ある音楽家が安心できる場にするよう工夫し、何が起こるかを注視していきましょう。

以上は、まさにソフトウェアのチームに必要なアドバイスです†。熟練開発者にふさわしい環境を提供する、という考え方が非常に重要なのです。

技能レベルが最高の開発者の生産性が、最低レベルの開発者と比べて桁違いに高いとすれば、現在の一般的な開発者対象の給与システムはまったく不合理と言わざるを得ません。何年も前の看護師と同様に我々も、かなりの量の専門的知識や技能を管理職や競争相手、その他の分野に流出してしまうという危険に絶えず直面していることになります。

最近ではより人件費の安い国への開発のアウトソーシングと海外委託が増加しているため、この傾向に拍車がかかり「プログラミングは単なる機械的な作業であり、最低入札者に注文すればよい」という考えを人々の思考に植え付けてしまっています。もちろん、それでうまくいくわけはないのです。

看護師と同様に、プログラミングの達人はプログラミングを続け、そこに有意義で報酬面でも満足の得られるキャリアを見つけ出せなくてはなりません。それを実現する第一歩は、トップクラスのプログラマーが組織にもたらす価値を反映した給与体系と昇進の道を設定することです。

† ドラマーの話はそのまま適用するには難しいアドバイスになっていますが、『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』[SH06]では開発プロジェクトの「リズム」について詳しく論じています。

ヒント5
達人であり続けるには、実践を続けること

1.5 道具のワナに注意

ソフトウェア開発におけるツールや形式モデル、モデリングなどの役割については、これまで数多くの本が書かれてきました。多数の人々がUMLとモデル駆動型アーキテクチャ（MDA）に将来性があると主張していますが、RUPとCMMプロセスモデルが業界の救世主となる、と主張していた人も同じくらい大勢いました。

けれども、あらゆる特効薬的なシナリオと同様に、どれを採用したところでそれほど簡単にはいかないと人々はすぐに気づきました。こうしたツールやモデルにはふさわしい場所があり、適切な環境では有用になり得ますが、ツールもモデルも望まれたような普遍的な万能薬にはなりませんでした。さらに悪いことには、こうしたアプローチの誤用により、利益よりも害の方が多かったのではないかと思われます。

モデルは道具であり、目的ではない

非常に興味深いことに、道具（ツール）や形式的なモデルの使用に関しては看護師にも同じような問題がありました。多数のアーキテクトやデザイナーがはまる同じワナに、看護師もはまっていたのです。モデルは最終的な目的ではなく、あくまでも道具だということを忘れる、というワナです。

ルールは与えられた状況で採用すべき最適な行動や道筋を教えてはくれません。モデルはせいぜい自転車の「補助輪」であり、進み出すための手助け（足助け）にはなりますが、その後はパフォーマンスのブレーキとなり、むしろ弊害をもたらします。

デボラ・ゴードンがベナーの本にひとつの章を寄稿していますが、その中でゴードンは、看護職において形式的モデルに過度に依存する危険性について概説しています。以下に、プログラマーという職業の特徴に合わせてゴードンの意見を再解釈してみましたが、元のゴードンの意見でさえ、とても身近なものに聞こえると思います。

モデルを現実と混同

モデルは現実ではないにも関わらず、簡単に両者を混同してしまいます。年若いプロジェクトマネージャーに関する語り草があります。プロジェクトのシニアプログラマーが自分の妊娠をプロジェクトマネージャーに告げ、プロジェクト期間中に出産する予定であると伝えたところ、マネージャーは妊娠や出産は「プロジェクト計画に入っていないじゃないか」と抗議したというのです。

形式化できない特徴を低く評価

プログラマーにとって問題解決の技能は非常に重要ですが、問題の解決方法を形式化することは非常に難しいものです。たとえば、ある問題が与えられたときに、それを解くのに椅子に座って何分考える必要があるかわかりますか？10分？ 1日？ 1週間？ 人によっても状態によっても変わってくるはずです。創造性や発明といったものはタイムレコーダーでは管理できませんし、達人が見つけ出す究極のテクニックをあらかじめ文書化しておくことなど不可能なのです。マネージャーがこのようなことをチームに求めたとしても、単に形式化が不可能というそれだけの理由で、経営陣は評価してくれない可能性が高いでしょう。

個人の自主性を否定する行動を許可

キーボードを闇雲に叩いてコードを粗製濫造する粗悪なプログラマーが欲しいと思っている人は誰もいないでしょう。思慮深く、責任感を持った開発者が欲しいのです。形式的なモデルに過度に依存すると、集団的な行動が評価され、個人の創造性は軽視される傾向になります†。

初心者に味方して、経験豊富な熟練者を疎外

これは特に危険な副作用です。方法論の対象を初心者に定めると、経験豊かなチームメンバーは劣悪な環境で仕事しなければならなくなるので、所属するチームもしくは組織を去ってしまうのです。

説明が細かすぎる

説明が細かすぎると収拾がつかなくなることがあり、先にこの章で説明した「無限後退」につながります。仮定する事柄を明示すると、そうした仮定をするために必要な仮定が露わになり、これが際限なく続きます。

複雑な状況の過度の単純化

ソフトウェア開発プロセスのフレームワークであるRUPの初期の提案者（と

† もちろん、バランスが必要です。チームや常識を無視して独立独行するような「無謀なプログラマー」はごめんでしょう。

最近の何人かの信奉者）は「プロセスに従うだけ」という概念に固執しています。エクストリームプログラミングの提唱者の中には、「この 12 個の――いや、ちょっと待て、たぶん 13 個だ――の慣行に従うだけですべてうまくいく」と強く主張する人たちがいます。どちらも正しくありません。いかなるプロジェクト、いかなる状況も、それよりずっと複雑です。「必要なのは……だけ」や「これさえすれば……」と話を切り出す人がいたら、その人は間違っている可能性が高いと思ってください。

極端な画一化の要求

同じ標準が一様にすべての状況に常に当てはまるとは限りません。直近のプロジェクトでうまくいったことが現在のプロジェクトでは大失敗をもたらすかもしれません。ボブとアリスが開発環境として Eclipse を使用してとても生産性が高くても、キャロルとテッドは沈没してしまうかもしれません。キャロルとテッドは開発環境として IntelliJ、プログラムの編集をするのに vi や Mac OS 用のエディタ TextMate を使いたいかもしれないのです[†]。

コンテキストの微妙な違いに対する配慮不足

形式的な手法は特定の用途向けではなく、典型的な用途向けに作られています。しかし「典型」が実際に発生することなどあるのでしょうか。コンテキストは達人のパフォーマンスにとって重要なものなのですが、形式的な手法は、定式化される過程において微妙な違いをすべて失ってしまうのが普通です（そうでなければならないはずです。さもなければ、朝、自分が飲むコーヒーを手に入れる方法を説明するためだけでも、何千ページもの説明が必要になってしまいます）。

ルールに従うべき時とそれを破るべき時の判断の誤り

ルールを破ってよいのはいつでしょうか？ いつでも破ってよいのでしょうか？ 決して破ってはいけないのでしょうか？ その間のどこかでしょうか？ どうやったらわかるのでしょうか？

標語化

言葉の標語化（スローガン化）が著しく進むことにより、それが自明なものとなり、やがては完全に意味を失ってしまいます。たとえば「我々は顧客中心の組織です！」といったスローガンがこの例です。アジャイルな手法が実効性を失ってしまうのは、まさしくこの問題を抱え込んだときなのです。

† ちなみに、私はこの本の執筆に、プログラマー用のエディタである vi、XEmacs（の vi モード）、そして TextMate を使用しました。

形式的な手法にはそれなりの長所も利用法もありますが、現在検討しているような目標の達成に役に立ちません。基準となるルールの確立は低い技能レベルの者にとっては有用かもしれませんが、ルールは判断の代用にはなりません。判断力が向上するにつれて、ルールへの依存の度合いをゆるめ、それとともに、制度上の強制も少なくする方向に移行していく必要があるのです。

ヒント6
創造性や直感力、創作力が必要なら、形式的な手法は避けること

ツールやモデルの持つ「偽りの影響力」に屈しないでください。思考に取って代わるものなど存在しません。

1.6　コンテキスト再考

ドレイファスモデルにおいて非常に重要なのは、「ビギナーにはコンテキストに依存しないルールが必要だが、達人はコンテキストに依存した直感を使う」という認識です。

> ホルマリン漬けの魚を持った男は真実をひとつ記し、多数の嘘を記録した。魚の本当の色はそんな色ではなく、感触も違い、そんな風に死んだわけではなく、そんな臭いもしなかった。
>
> ——ジョン・スタインベック著『コルテスの海』

『コルテスの海』でスタインベックは、コンテキストと真実について考察しています。パシフィックシエラという名前の魚を研究室で記述するのは簡単です。「ひどい臭いのするビンを開け、硬直した無色の魚をホルマリン溶液から取り出し、とげの本数を数え、真実である『D. XVII-l5-IX』を書き記す」だけでよいのです。それは科学的な真実ですが、コンテキストを欠いています。「玉虫色に光り、尾が空中でむち打つ」生きている魚とは違います。生息環境で生きている魚は、研究室でビンに保存された魚とは基本的に別物です。コンテキストが重要なのです。

高収入のコンサルタントが「場合によりけりです」という答えを好んで使うことをご存じでしょうか。もちろんコンサルタントの言うとおりです。コンサルタントの分析は驚くほど多数のパラメータに左右されます。無関係な詳細は無視しますが、調査を必要とする意味のあるあらゆる事項が、達人コンサルタントにはわかっ

ているのです。コンテキストが重要なのです。

錠のかかったドアを達人に開けてもらう場合を考えてみましょう。たとえば、火事になっている家のドアを開けて中にいる赤ちゃんを助け出すのと、ウォーターゲートホテルで何の痕跡も残さずにピッキングをするのはまったく別の行為です。コンテキストが重要なのです[†]。

コンテキストから切り離された客観性に注意

コンテキストから切り離された客観性――すなわちコンテキストから切り離した後に客観的になろうとすること――には危険がつきものです。たとえば前述のスタインベックの引用に出てくる、恐らくは研究目的で解剖された魚は、逆巻く波を滑るように泳ぐ銀色に輝く動物とはまったくの別物です。

鍵開けの例では、「錠がかかっているこのドアを開けたい」と言うだけでは十分ではないのです。どのような状況（コンテキスト）なのかが重要なのです。ドアを開けなければならない理由は？　斧やチェーンソー、ピッキングの道具を使ってもいいのか？　それとも、単純に裏へ回って裏口から入ることはできないのか？

システム思考（システムズシンキング）では、オブジェクト指向プログラミングと同様に、オブジェクト自体が意味を持つというよりは、オブジェクトとオブジェクトとの関係が重要な意味を持つことが多いのです。こうした関係がコンテキストの形成に寄与することになり、そしてその結果形成されたコンテキストが違いを生み出します。

コンテキストは重要ですが、ドレイファスモデルの下位レベルの者にはこの重要性がわかりません。したがって、ドレイファスモデルの「ハシゴ」を登る方法を考えなければなりません。

1.7　日々のドレイファスモデル

さて、ここまでの話は興味深いものだったでしょうか？　そうであるとよいと思いますが、今度はよりプラグマティックな面に目を向けていきましょう。実際のところ、ドレイファスモデルは何がよいのでしょうか。ドレイファスモデルの知識を身につけ、それで何ができるのでしょう。どのように利用すれば自分のためになるのでしょうか。

† ピッキングに関する専門的な話題は『*How to Open Locks with Improvised Tools*』[Con01]を参照。

フリーサイズは存在しない

まず覚えておいて欲しいのが、誰にでもピッタリ合う「フリーサイズ」は存在しないということです。先に見たように、人によって何が必要かは変わってきます。その人の成長にとって何が必要かは、時とともに変化します。そして、チームの他のメンバーの意見をどう聞き、どのように対応するかも、意見をくれたメンバーの技能レベルを考慮して決める必要があるのです。

初心者には小さな成功とコンテキストに左右されないルールが必要です。未経験の状況で適切な判断が下せると期待してはなりません。問題が与えられると、初心者は関係がありそうなことを、個々の重要性は無視して何から何まで考慮して、その問題を解こうとします。自分をシステムの一部とは思っていないので、自らが全体に与える影響に気づきません。初心者にとって必要な手助けをすることは大切です。しかし、まだ周囲の見えていない者に全体像を示して、無用な混乱を招くようなことは避けなければなりません。

これに対して、初心者から見てドレイファスモデルの反対側の端に位置する達人は、全体像が見えていなければなりません。達人の判断力という恩恵を享受したければ、達人の手足を縛るようなことをしてはなりません。判断力に取って代わることを目的とした制約が強く官僚的なルールを、使ってはならないのです。達人は、良かれ悪しかれ、自身をシステムそのものの一部と考えており、一般人が考えるよりも個人的な嗜好を伴って物事を受け止める傾向があるということを覚えておいてください。

チームにはさまざまな技能レベルの人がいることが理想です。達人ばかりのチームにはそれなりの困難が伴います。皆が森林について思い巡らす傍らで、一本一本の木について心配する人も必要です。

ドレイファスモデルについてはなじみのない人が多いでしょうから、このモデルの理解と利用に関して、ほとんどの読者は「初心者」ということになります。ドレイファスモデルの理解それ自体がひとつの技能であり、したがってその学び方を学ぶこと自体がドレイファスモデルの応用対象となります。

ヒント7
習得するという技能を習得する

次章以降の道しるべ

この本のこれ以降の部分では、ここまでで説明してきたようなドレイファスモデルの教えを指針として使います。達人に通ずる道を歩み始めるためには、以下が必要です。

- さらなる直感力の育成
- コンテキストの重要性の認識、および状況のパターンに対する重要性の認識
- 自分自身の経験のさらなる活用

このような目標をどのように達成できるのかを見るために、次の章では脳がどのように働くのかを詳しく見ることから始めましょう。

さっそく実行 ⬇

- □ 自分自身を評価してみてください。仕事で主に使う技能について、自分はドレイファスモデルのどのレベルに位置しますか。現在の技能レベルが自分にどのようなインパクトを与えているかを列挙してください。
- □ 上にあげた以外の技能——仕事では（ほとんど）使わない技能——について、ドレイファスモデルのどの技能レベルに位置するかを考えてください。この評価を行うときには、二次無能力の可能性に注意してください。
- □ 上であげた各技能について、次のレベルに進むために何が必要かを判断してください。そして、この本のこれ以降の部分を読み進める際に、それを頭の片隅に置いておいてください。
- □ プロジェクトチームに所属していたときに経験した問題を振り返ってみましょう。チームがドレイファスモデルのことを知っていたら、そうした問題は回避できたでしょうか？ 違う道をたどるために別の方法はあるでしょうか？
- □ チームメイトのことを考えてみましょう。ドレイファスモデルのどの段階にいるでしょうか？ それは自分にとってどのように役立つでしょうか？