第3章
プロジェクト・ダイナミクス

人間の活動には、期末レポートの作成から原子力発電所の建設に至るまで、驚くほど類似したダイナミクスを見出すことができます。プロジェクトは、まず目標設定があり、次にその達成に向けた多くの作業が続き、最終的に目標の達成を目指します。しかし実際は目標が未達成に終わり、予算超過、納期遅れ、品質低下といった問題が発生することも少なくありません。

ここでは**プロジェクト実行のプロセスを理解するためのモデル**を開発します。このモデルは、新製品の設計、プレゼンテーションの準備、ソフトウェア開発、スペースシャトル建造等、さまざまな活動に適用できる汎用性の高いものです。ここでは分かりやすさを重視し新しいビルの設計を例に話を進めます。

モデルの概念化と作成に**イテレーティブ（反復的）なアプローチ**を採用します。これは最も単純な構造から始め、段階的にモデルを改善していく手法です。このアプローチは大規模で複雑なモデルを作成したものの、シミュレーション結果が意味をなさないといった事態を防ぐ上で非常に有効です。イテレーティブな開発ではモデルに変更を加えるたびにシミュレーションを実行し、追加した部分の効果を観察・確認することが重要です。

モデル開発は構造変更とそのシミュレーション結果に基づいて次のステップを決めるため、必然的にコンピュータを使った作業となります。コンピュータはこのような作業に適していますが、使いこなすためには**作業の目的や課題を常に意識することが更に重要**です。シミュレーションを実行する際は、どのような結果が予測されるかをあらかじめ自問自答しておくべきです。もし予期せぬ結果が出た場合は、なぜそうなったのかを必ず考察し、正しいと思われる結果が出た場合でも、それが正しいと考えられる理由を確認するようにしてください。客観性を保つためシミュレーション結果に関する予測を事前に書き留めておくことをお勧めします。経験豊富なモデル開発者の多くは、実行時に発見した意外な振る舞いや、それに対する洞察をノートに記録しています。

3.1. **タスクの完了**（project1.mdl）

あらゆるプロジェクトの最も基本的な特徴は何らかのすべきことがあり、少なくとも部分的には実行されるという点です。これをモデル化するために図 26 プロジェクトのストック・フロー図に示すように 2 つのストックとその間のフローから始めます。

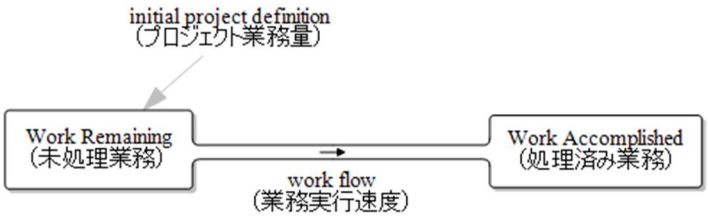


図 *26* プロジェクトのストック・フロー図

initial project definition(プロジェクト業務量)を 1000 と設定します。このプロジェクトは、10 ヶ月で完了するプロジェクトであるとします。完了時間を確認するために十分な余裕を持たせてプロジェクトを 24 ヶ月間実行してみます。その際 **TIME STEP** を **0.0625** とします。 これは 1 ヶ月の 16 分の 1 の刻みで計算するという意味で、プロジェクト活動の変化を捉えるには十分な精度です。方程式は次のとおりです。

Work Remaining = INTEG (-work flow, initial project definition)
Units: Drawing
Work Accomplished = INTEG (work flow, 0)
Units: Drawing
initial project definition = 1000
Units: Drawing
Work flow = 100
Units: Drawing/Month

ここで使われている単位の **Drawing（設計図面）**はビルの設計の場合等を想定したものです。その他のケースでは、**Task(タスク)**、**Line(ソフトウェアコードの行数)**等、他の単位も自由に使うことができます。ここではプロジェクト全体を一つのまとまりとして扱いますが、実際にはプロジェクトは全てが同じ大きさの仕事の集合ではなく、大きいものや小さいものが組み合わされています。式のうえで、**一つひとつの仕事は平均を表している**とします。実際の問題に適用する際に、このことを心に留めておくことは**物事の本質をシンプルに表現するという意味で非常に重要なこと**です。

このモデルのシミュレーションを実行すると、図 27 プロジェクト遂行のシミュレーション結果のようなグラフになります。

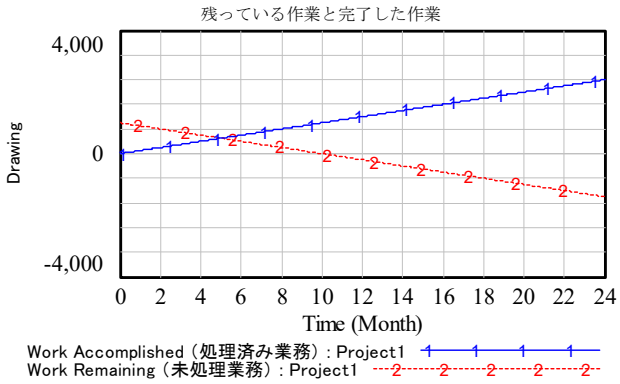


図 27 プロジェクト遂行のシミュレーション結果

10 ヶ月後に **Work Remaining(未処理業務)**は 0 になります。しかし、このシミュレーションでは **Work Remaining(未処理業務)**は減り続けます。これはプロジェクトを停止させるためのロジックが組み込まれていないためです。

3.2. 作業の停止(project2.mdl)

プロジェクトモデルを停止させる方法は二つあります。

一つ目はプロジェクトが終了する時点で単純に**シミュレーション自体を止める**方法です。これは **FINAL TIME（最終時間）**を決める式を指定することで実現できます。

ここで採用したアプローチは、もう一つの方法です。それはプロジェクトが完了した時点で活動を停止するという考え方です。このために **project is done(プロジェクト完了フラグ)**という概念を追加し、**project is done(プロジェクト完了フラグ)**から **work flow(業務実行速度)**までを矢印で接続します。その様子を図 28 プロジェクト完了フラグを導入として示します。

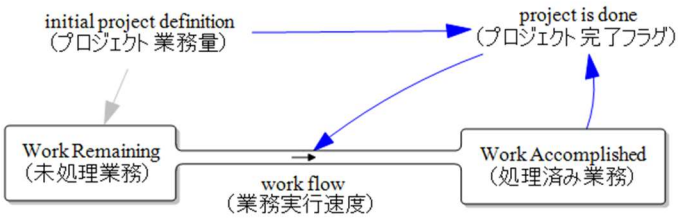


図 28 プロジェクト完了フラグを導入

project is done(プロジェクト完了フラグ)が **Work Remaining(残りの作業量)**ではなく **Work Accomplished(処理済み業務)**に依存するようにすることで、スケジュールや予算上の問題が発生した場合に、プロジェクトの範囲を縮小するという判断をくだすために有効なメカニズムとなります。

これを実現するために、次の式を追加します。

project is done = IF THEN ELSE (
Work Accomplished >= initial project definition, 1, 0
)
Units: Dmnl
work flow = IF THEN ELSE (
project is done, 0, 100
)
Units: Drawing/Month

このシミュレーション結果は図 29 プロジェクト完了フラグを導入後のシミュレーション結果のようになります。

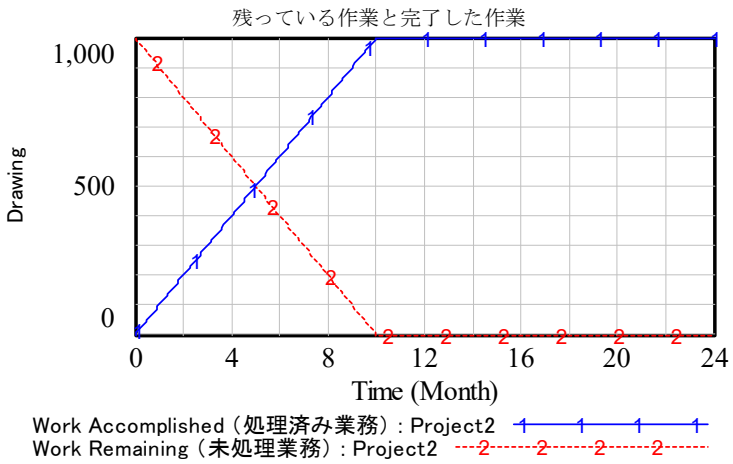


図 29 プロジェクト完了フラグを導入後のシミュレーション結果

3.2.1. 積分法

次に進む前に積分法に関する注意点を説明します。このモデルには作業が急に停止する部分があります。このようなプロジェクトモデルにはオイラー積分法が適していますが、通常ある種のオーバーシュートが発生します。例えば作業量が 101%まで達してしまうような現象です。このモデルでもその様子を後ほど観察できます。

このオーバーシュートはモデルの概念としては重要ではありませんし、実用上の問題にもなりません。しかし見た目を改善するために修正したいと考える人もいます。本章では細かな計算よりも、より重要な概念上の問題に焦点を当てます。積分法に関する詳細は第8章を参照してください。

3.3. エラーとリワーク(project3.mdl)

これまでのところ実行される作業にエラーはないと仮定してきました。しかし現実には、この前提は当てはまりません。エラーは人為的な問題、技術的な見落とし、うっかりミス、あるいはコミュニケーション不足等さまざまな原因で発生します。さらにエラーが発生しても、すぐに発見されるわけではありません。多くの場合、エラーはレビューや統合作業で明らかになるまで発見されません。

そこでここでは業務が Work Remaining(未処理業務)から work flow(業務実行速度)のフローで Work Accomplished(処理済業務)に向かうのと同時に、一定の割合すなわち（**1-歩留まり**）でエラーを含む仕事が **Undiscovered Rework(未発見の不良業務)**に蓄積されていくと仮定します。そして一定の時定数 **time to detect errors(不良検出時間)**でエラーが発見されていき、それと同じ速度で **Work Accomplished(処理済み業務)**から **Work Remining(未処理業務)**に業務を戻します。以上のアイデアを構造に落とし込んだスケッチが図 30 リワークを組み込んだモデルです。

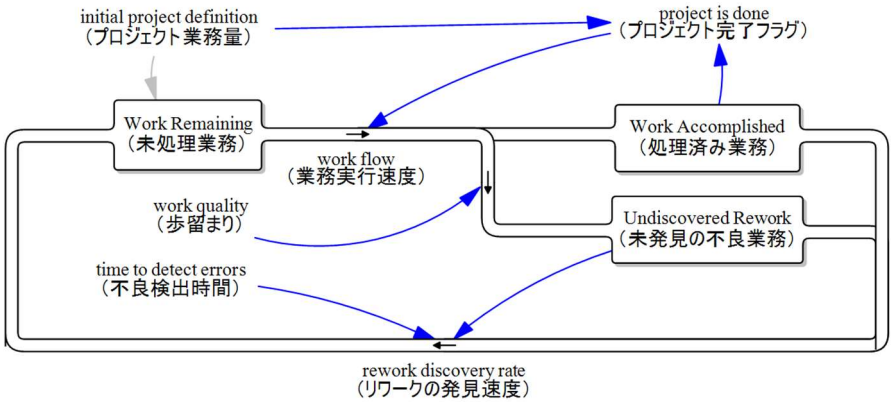


図 30 リワークを組み込んだモデル

別の表現方法としては、作業のフローを「うまくいった作業」と「うまくいかなかった作業」に分け、「うまくいかなかった作業」のフローを「なされるべき作業」に戻すという方法があります。もしこの方法を採用する場合、**Work Accomplished(完了した作業)**

という概念は、「うまくいった仕事」と、「うまくいかなかった作業」の合計として再定義する必要があります。

ワンポイントアドバイス

フローを表すパイプを分岐させる書式の読み方ですが、図 30 リワークを組み込んだモデル中央部分にある分岐したパイプは、**Work Remaining(未処理業務)**から**work flow(業務実行速度)**で決定されるフローで**Work Accomplished(処理済み業務)**に至るフローと、**Work Remaining(未処理業務)**から**work flow(業務実行速度)**と **work quality(歩留まり)**から決定されるフローを経て **Undiscovered Rework(未発見の不良業務)**に至るフローを併合して表現したものです。同様に、図 30 リワークを組み込んだモデルの下部にある合流したパイプは、**Work Accomplished(処理済み業務)**から **rework discovery rate(リワークの発見速度)**で決定されるフローで**Work Remaining(未処理業務)**に至るフローと、**Undiscovered Rework(未発見の不良業務)**から**rework discovery rate(リワークの発見速度)**で決定されるフローで **Work Remaining(未処理業務)**に至るフローを併合して表現したものです。なれれば大変スマートな表記方法です。

3.3.1. ダイヤグラムの作図

上記図 30 リワークを組み込んだモデルのダイヤグラムを描くための手順を説明します。まずはストックとフローだけを作成します。フローツールを選択し**Work Remaining(未処理業務)**から**Work Accomplished(処理済み業務)**に**work flow(業務実行速度)**と名付けられたバルブを描きます。次に雲から**Undiscovered Rework(未発見の不良業務)**に流入する名前のないバルブを描きます。名前のないバルブはフロー名の入力を求められたときエスケープキーを押下することで作成できます。次に **work flow(業務実行速度)**と名付けられたバルブを最初に置かれた中央付近から左側に少し寄せておく必要があります。次に**Undiscovered Rework(未発見の不良業務)**にインフローしているフローの雲を消去します。その際に雲とともに名前のないバルブより左にある部分のパイプが消えます。その様子を図 31 ストックとフローのみを作成に示します。

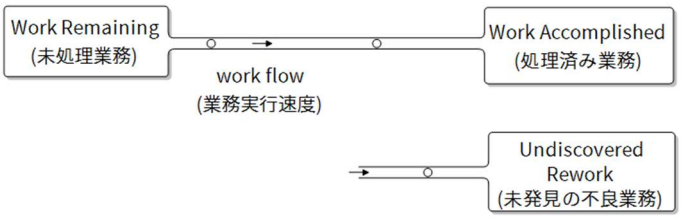


図 *31* ストックとフローのみを作成

次に、フローツールを選択し **work flow(業務実行速度)**と名付けられたバルブをクリックします。次にシフトキーを押して、シフトキーを押したままマウスを右に動かし、そしてシフトキーを押したままもう一度クリックし、こんどはマウスをまっすぐ垂直な位置に動かし、シフトキーを押したままにして、さらにもう一度クリックします。最後にシフトキーを放し、**Undiscovered Rework(未発見の不良業務)**にインフローする名前のないバルブをクリックします。この際にバルブ名の入力を求められますが、名前のないバルブを作成するためにエスケープキーを押下します。そうすると、図 32 業務遂行速度から分岐したパイプを作図のようになるはずです。必要により 2 つのバルブの相対的な配置を少し手直しが必要かもしれません。これらの手順で描いた後の様子を図 32 業務遂行速度から分岐したパイプを作図に示します。

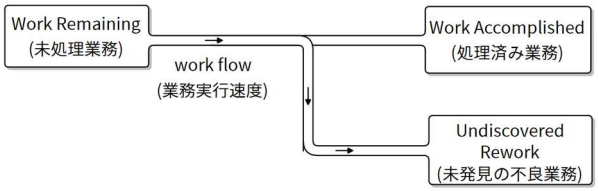


図 *32* 業務遂行速度から分岐したパイプを作図

次に、**Undiscovered Rework(未発見の不良業務)**から**残っている作業(Work remaining)**へフローツールを使って線を引きます。そのためには、まず**Undiscovered Rework(未発見の不良業務)**をクリックし、それからシフトキーを押します。シフトキーを押したまま、右にまっすぐ動かしてクリックします。それから下にまっすぐに線を引いてクリックします。次にシフトキーを押したまま左に動かして線を引きクリックします。シフトキーは押したまま上に動かしてクリックします。シフトキーはそのままにして右に動かし、最後にシフトキーを放して**Work Remaining(残っている作業)**をクリックします。バルブ名の入力を促されたら、**rework discovery rate(リワークの発見速度)**という名前をバルブにつけます。その様子を図 33　リワークの発見のフローを作図に示します。

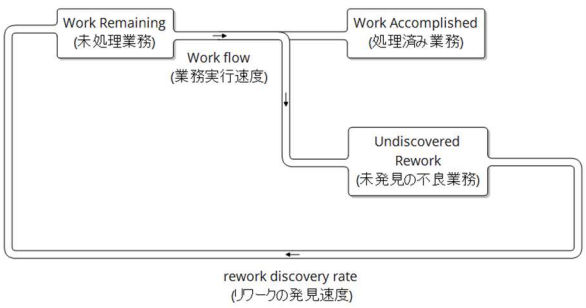


図 *33*　リワークの発見のフローを作図

Work Accomplished(処理済み業務)から始まる最後のパイプを作成するには、フローツールを使用してください。既存のパイプが回り込んでいるのと同じ水平位置まで右に移動し、シフトキーを押したままクリックします。既存のパイプの底の位置まで移動し、シフトキーを押したままクリックします。最後にシフトキーを放し、**rework discovery rate(リワークの発見速度)**のバルブをクリックします。

必要に応じて「スケッチ移動」ツールを選択し、スケッチの要素を適切に配置するために調整する必要があるかもしれません。以上の手順により図 30 リワークを組み込んだモデルに示すダイヤグラムの基本部分を描くことができます。

ワンポイントアドバイス

モデリングガイドの中で紹介されているリワークのモデルのスケッチに用いられているフローパイプの分岐は、スマートな表現ですので和訳版のなかでもそのまま使っていきます。ここではリワークのモデリングについての理解を深めるために、あえてフローパイプの分岐を用いない別の表現でリワークの考え方を説明します。

モデリングガイドの中でも、リワークのモデルの考え方に二通りの方法があることが紹介されています。一つ目は業務にエラーが混入しているか否かに関わらず、プロジェクトの業務は粛々と **Work Remaining(未処理業務)**が**処理済み業務(処理済み業務)**に **work flow(業務実行速度)**でフローしてゆくという考え方です。業務を終えた時点では、終えた業務の中にエラーがあったり、将来不具合を発生させたりする業務は判別できません。しかしながら、この業務では１万人・日[4]に1件不具合が発生する、１００作業に1作業は不具合を発生させる等の認識された品質に関する情報は把握できます。そのような品質の状況に応じた**error occurrence rate(エラー発生速度)**で **Undiscovered Rework(未発見の不良業務)**が蓄積され、それが**rework discovery rate(リワークの発見速度)**でアウトフローしてゆき、そのアウトフローと同じ速度で業務が **Work Accomplished(処理済み業務)**から **Work Remaining(未処理業務)**に戻されます。この構造をスケッチすれば図 34 未発見の不良業務に同期させて処理済業務を未処理業務に戻すのようになります。あくまでも **Work Remaining(未処理業務)**に戻されるのは **Work Accomplished(処理済み業務)**であり、**Undiscoverd Rework(未発見の不良業務)**からのアウトフローは**rework rate(リワークの速度)**を決定するための原因変数として用いられていることを確認してください。

[4] 人・日はニンニチと読み、作業の量を表す単位です。例えば、３人が１０日間作業して終えることができる業務量は３０人・日といった使い方をします。

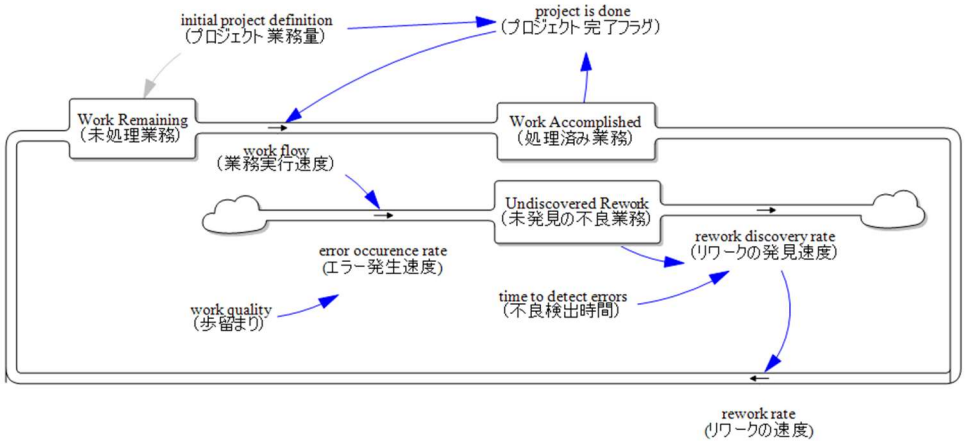


図 *34* 未発見の不良業務に同期させて処理済業務を未処理業務に戻す方法

二つ目は、**Work Remaining(未処理業務)**からのアウトフローを、**successful work flow(上手くいった業務実行速度)**のフローを経て **Successful Work(上手くいった業務)**へ蓄積される業務と**unsuccessful work rate(上手くいかなかった業務速度)**を経て**Unsuccessful Work(上手くいかなかった業務)**へ蓄積される業務に分ける方法です。そして、**Unsuccessful Work(上手くいかなかった業務)**が**time to rework(リワーク所要時間)**の時定数で**Work Remaining(未処理業務)**に戻されます。ここで注意すべきなのは、一つ目の方法で考えたストック**Work Accomplished(処理済の業務)**に対応する量は、二つ目の方法の**Successful Work(上手くいった業務)**と**Unsuccessful Work(上手くいかなかった業務)**の和になります。上手くいった業務と上手くいかなかった業務が判明するのは、業務を終えてから一定の時間が経過した後であったり、プロジェクトが完了した後であったりします。業務を終えた時点では個別具体的には終えた業務がいずれに属するのかは不明です。ただし認識した品質情報に応じて、確率・統計的に、上手くいった業務遂行速度と上手くいかなかった業務遂行速度のフローを計算することは可能です。この構造をスケッチすれば図 35 上手くいった業務と上手くいかなかった業務に分ける方法のようになります。

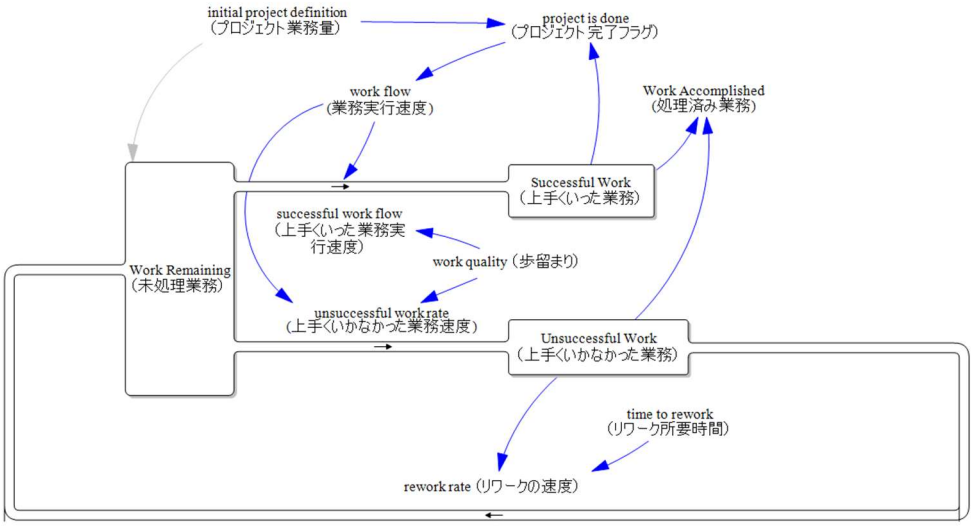


図 *35* 上手くいった業務と上手くいかなかった業務に分ける方法

3.3.2. エラーに関する処理の統合

それでは本論に戻りましょう。図 30 リワークを組み込んだモデルに示すとおりダイヤグラムを変更し、次のように式を変更・追加します。

rework discovery rate = Undiscovered Rework / time to detect errors
Units: Drawing/Month
time to detect errors = 3
Units: Month

rework discovery rate(リワークの発見速度)は、エラーの種類（例えば、再現性の有無等）や発見者が誰なのか（開発者なのかユーザーなのか等）によって変わります。この時間はプロジェクトの段階によっても変わるためプロジェクト全体で一定ではありません。その詳細については後で説明します。ここでは**平均すると 3 か月程度**とします。

```
Undiscovered Rework = INTEG (
    work flow*(1-WORK QUALITY) - rework discovery rate, 0
    )
Units: Drawing
Work Accomplished = INTEG (
    work flow - rework discovery rate, 0
    )
Units: Drawing
WORK QUALITY = 0.9
Units: Dmnl
Work Remaining = INTEG (
    rework discovery rate - work flow, initial project definition
    )
Units: Drawing
```

これらを付け加えると振る舞いは図 36 project3.mdl のシミュレーション結果のようになります。

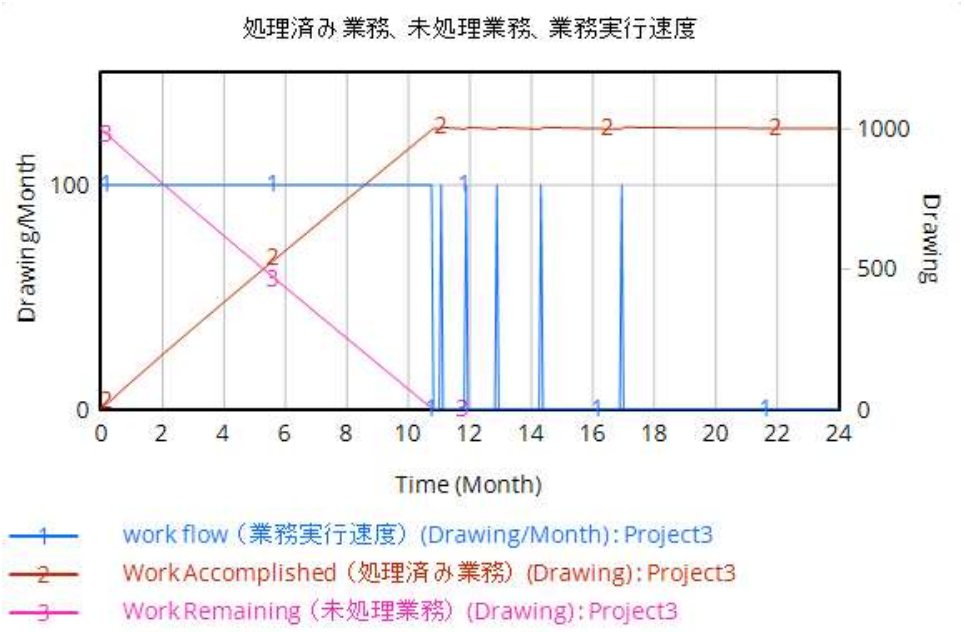


図*36 project3.mdl* のシミュレーション結果

ここで 2 つのことに気づきます。一つ目はすべての作業が正確に行われるとは限らないためプロジェクトは少し長くかかります。二つ目はプロジェクト終了後、未発見のリワークが明らかになりプロジェクトが何度も再開される点です。

3.4. **リワークの発見**(project4.mdl)

Project3.mdl ではプロジェクトの終了時点で **Undiscovered Rework(未発見の不良業務)**がピークに達するため、**rework discovery rate(リワーク発見速度)**もまたその時点でピークとなります。その様子を図 37 未発見の不良業務とリワークの発見速度に示します。

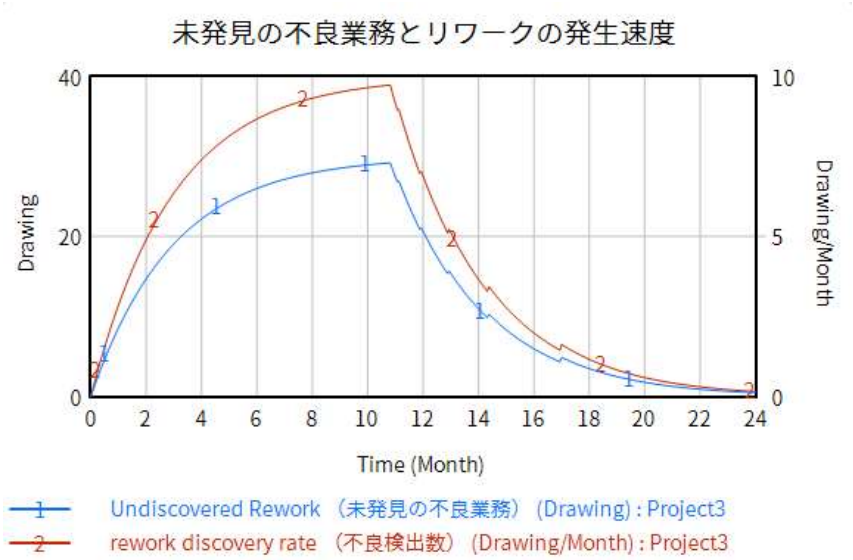


図 37 未発見の不良業務とリワークの発見速度

Undiscovered Rework(未発見の不良業務)は本質的に観察だけでは発見できないため、プロジェクトの最終段階でリワークの発見量が増加することになります。これはパズルで最後のピースを置く際に、これまで見過ごされていた間違いや見落としが判明し、ピースの置き直しが多く発生することと非常によく似ています。例えば設備導入時に通路の幅や換気用ダクトの寸法が間違っていた、といったことが起こり得るのです。これらの概念を単純化して **time to detect errors(不良検出時間)**として定義します。ここでさらに変数 **fraction complete(進捗率)**を追加します。この変数はプロジェクトの度合いを表します。この新しい変数を使って **project is done(プロジェクト完了フラグ)**の式を修正します。変更された式は次のとおりです 。

fraction complete = Work Accomplished / initial project definition
Units: Dmnl
project is done = IF THEN ELSE (fraction complete >= 1, 1, 0)
Units: Dmnl
time to detect errors = time to detect error lookup (fraction complete)
Units: Month
time to detect error lookup ((0,5), (0.5,3), (1,0.5))
Units: Month

プロジェクトが終了に近づくにつれて **time to detect errors(不良検出時間)**が十分に小さな値になるように変更するため、**fraction complete(進捗率)**をパラメータとして表

関数**time to detect error lookup(不良検出時間表)**を使用します。Vensim では方程式ツールを用いて表関数の入力・出力特性を座標で定義することもできますし、同じ方程式ツールを用いてフラフ表示に基づいて表関数を定義したり調節したりすることもできます。**time to detect error lookup(不良検出時間表)**を方程式ツールでグラフ表示すると図 38 表関数 time to detect error lookup (不良検出時間表)の方程式ツールによる表示のようになります。

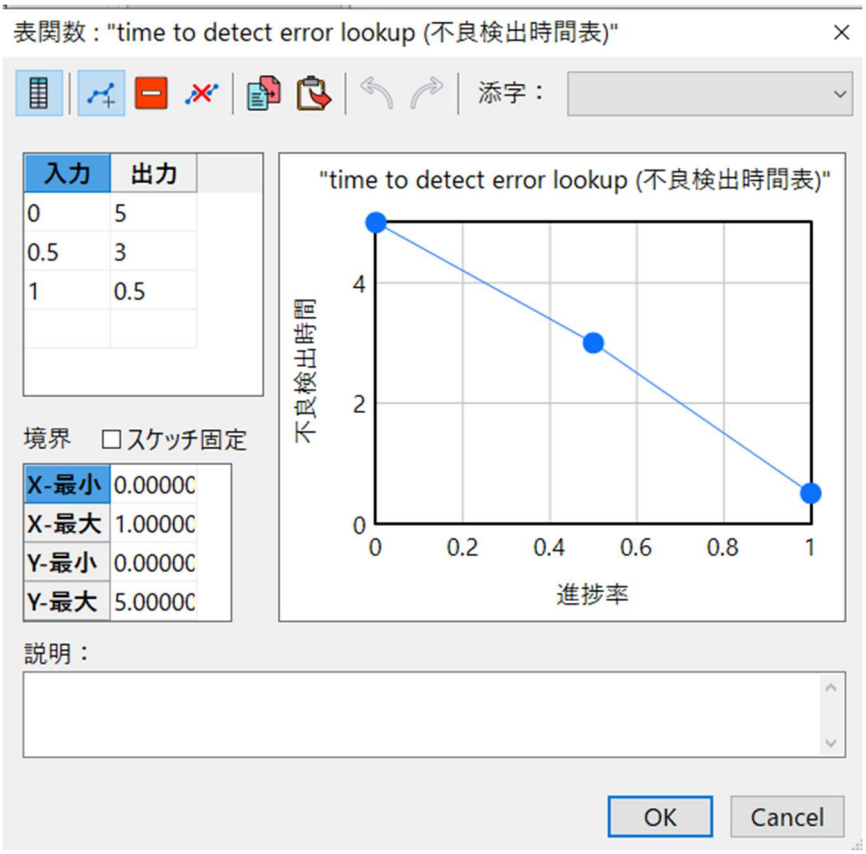


図 *38* 表関数 *time to detect error lookup (*不良検出時間表)の方程式ツールによる表示

これらを反映したモデルスケッチを**エラー! 参照元が見つかりません。** として示します。

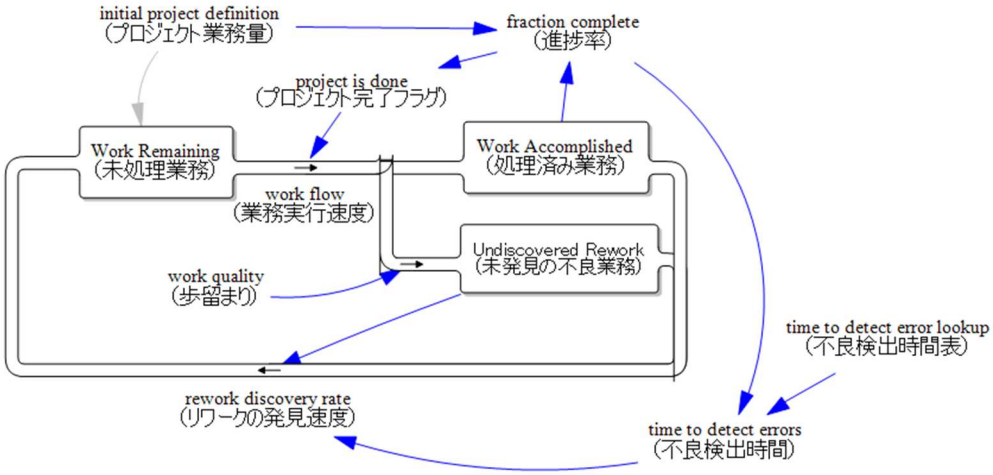


図 *39* 進捗率と不良検出時間表を導入したモデル*(project4.mdl)*

Project4.mld のシミュレーション結果を project3.mdl のシミュレーション結果と対照的に示したものが図 40 project4.mdl のシミュレーション結果（下図）

図 40project4.mdl のシミュレーション結果になります。このうち、上図は Undiscoverd Rework(未発見の不良業務)、中図は rework discovery rate(リワーク発見速度)、下図は work flow(業務実行速度)になります。

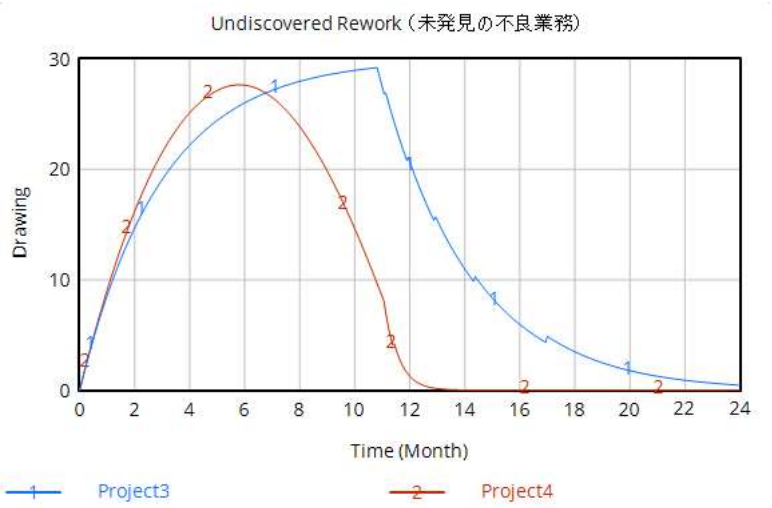


図 40 project4.mdl のシミュレーション結果（上図）

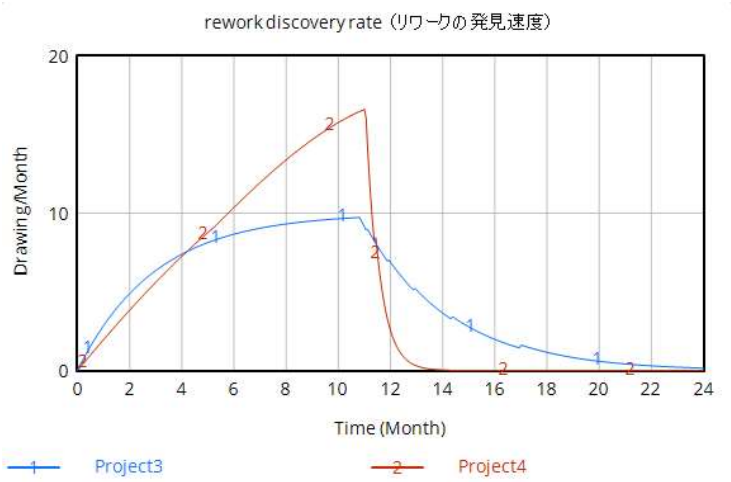


図 40 project4.mdl のシミュレーション結果（中図）

図 40 project4.mdl のシミュレーション結果（下図）

図 *40project4.mdl* のシミュレーション結果

未発見の不良業務（上図）、リワークの発見速度（中図）、業務実行速度（下図）

これらの結果から、プロジェクトの進捗率による不良検出時間の変化を考慮するとプロジェクトの完了はわずかに遅くなりますが、完了後のリワークがほとんど発生しなくなることが分かります。シミュレーションでは、**time to detect errors(不良検出時間)**の計算において、**time to detect error lookup(不良検出時間表)**に関するエラーメッセージが出ることがあります。これは、行われた仕事の量が、元の仕事量をわずかに超過することによって発生するものです。これは、プロジェクトを不連続に停止させたことが原因と考えられます。考え方としては問題ないため、今のところこのままとします。

3.5. スケジュール(project5.mdl)

これまでのところ、**work flow(業務実行速度)**は定数として扱ってきました。そのためスケジュールからのフィードバックもありませんでした。プロジェクト管理の目的はプロジェクトをスケジュールどおり進めることです。そのためにスケジュールに注意を払い、スケジュールに合わせてリソースを調整することが必要です。完了予定日があるならば、それに基づいて残された日数を計算できます。また残りの仕事の量が分かっているので、スケジュールに間に合わせるために、どれだけ仕事を速めなければならないかを決めることもできます。理解を促すため、先に業務実行速度の制御のための構造を図 411　業務実行速度の制御のための構造にもとづいて説明します。引き続き project4.mdl を修正して project5.mdl を作成します。その際にビュー画面１に変更なしの部分を、ビュー画面２に変更した部分を描くようにします。

ここでの課題は、**work flow(業務実行速度)**にスケジュールを反映することです。そのためにスケジュールを反映した業務実行速度という意味で変数**required work flow(要求実行速度)**を導入します。**required work flow(要求実行速度)**は**Work Remaining(未処理業務)**と**scheduled time remaining(残存期間)**から導かれます。**scheduled time remaining(残存期間)**は**scheduled completion date(完了期日)**から**Time**(現在時刻)を差し引いた**scheduled time remaining(残存期間)**から導かれます。なおVensimにおいて**Time**はシステム変数でシミュレーションにおける現在時刻を意味します。図 411　業務実行速度の制御のための構造は以上の関係を図示したものです。　　これをこのままビュー画面２に描くことはできません。これから変更すべき内容について考え方を理解するための図であることに注意してください。

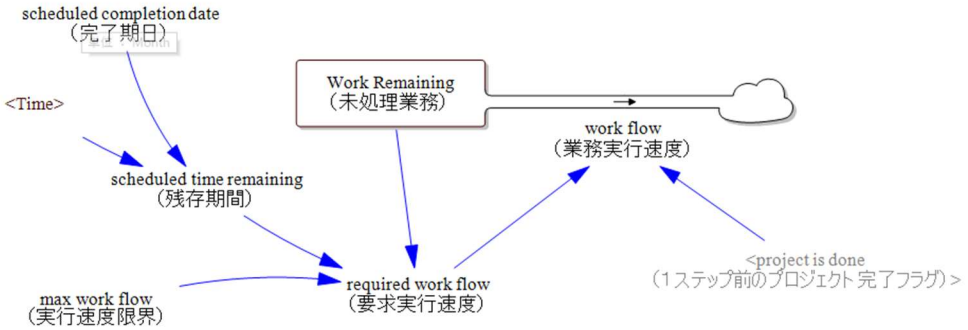


図 *411*　業務実行速度の制御のための構造

これらのことを実現するための式は次のようになります。

required work flow =MIN (
max work flow,
XIDZ (
Work Remaining,
scheduled time remaining,
max work flow
)
)
Units: Drawing/Month

ここでは、**MIN(最小値)**関数を使って、**required work flow(要求実行速度)**が**max work flow(実行速度限界)**よりも小さくなるように制約します。

XIDZ(X if Divide by Zero)関数は、もし 第 2 引数の **scheduled time remaining(残存期間)**がゼロでなければ、**Work Remaining（未処理業務）**を**scheduled time remaining(残存期間)**で除する機能を持ちます。0 で除することは意味がなく Vensim はエラーとなるので、今後このような場面でこの関数を使用します。

scheduled time remaining = MAX (0, scheduled completion date - Time)
Units: Month
scheduled completion date = 10
Units: Month
max work flow = 500
Units: Drawing/Month
work flow = IF THEN ELSE (project is done, 0, required work flow)
Units: Drawing/Month

ワンポイントアドバイス

図 411　業務実行速度の制御のための構造で示したアイデアに従って、実際に project4.mdl に変更を加えていきます。操作の詳細な説明はモデリングガイドの趣旨から外れるかもしれませんので、そのための操作をワンポイントアドバイスとして記します。まず、変更した部分の構造を格納するための新規のビューをつけ加えます。画面左下のビューの選択で「＊＊新規＊＊」を選択してください。そしてビューの名前を「2」に変更します。ビュー2に図 411　業務実行速度の制御のための構造に示したアイデアをスケッチしていきます。その結果が図 42 project5.mdl の変更部分（ビュー2）です。まず **required work flow(要求実行速度)**は新しい変数ですから、ビュー2 において変数ツールを使って定義します。**required work flow(要求実行速度)**を計算するための必要な3つの変数のうち、**Work Remaining(未処理業務)**はすでにビュー１の中で定義されています。このようなときは代行変数ツールを用いて、**Work**

Remaining(未処理業務)を選択します。これでビュー１の**Work Remaining(未処理業務)**をビュー2 の中で参照できるようになります。ただし、代行変数は参照だけで代入はできないので注意が必要です。代入が必要なときは、代行変数ツールの代わりにモデル変数ツールを用いてスケッチするか変数 **Work Remaining(未処理業務)**を定義したビュー１の中で **Work Remaining(未処理業務)**の式を変更する必要があります。**scheduled time remaining(残存期間)** と**max work flow(実行速度限界)**は新しい変数ですので、ビュー２の中で変数ツールを使って定義します。**scheduled time remaining(残存期間)**の計算に必要な２つの変数のうち、**Time** は Vensim のシステム変数であり、既に定義済みの変数です。よって代行変数ツールで **Time** を選択してビュー２の中で参照できるようにします。**scheduled completion date(完了期日)**は新しい変数ですので変数ツールを使ってビュー2 のなかで定義します。

ビュー１については、考え方の変化はありませんが、ビュー２の中で定義した**required work flow(要求実行速度)**と結合するための変更が必要です。その結果を図 43 project5.mll の変更なしの部分（ビュー1）に示します。まず**work flow(業務実行速度)**は原因変数として**required work flow(要求実行速度)**を参照しますから、ビュー１のなかで代行変数ツールを使って**required work flow(要求実行速度)**の代行変数をビュー１のなかでスケッチします。そのうえでビュー１にて代行変数**<required work flow(要求業務実行速度)>**から**work flow(業務実行速度)**に因果関係を示す矢印を引きます。最後に方程式ツールを使って**work flow(業務実行速度)の**式を定義すれば完了です。なお新しく定義した変数と**work flow(業務実行速度)**の式については前項の式の説明で示したとおり入力してください。

以上の式を反映したモデル project5.mdl を図 43 project5.mll の変更なしの部分（ビュー1）図 42 project5.mdl の変更部分（ビュー2）に示します。

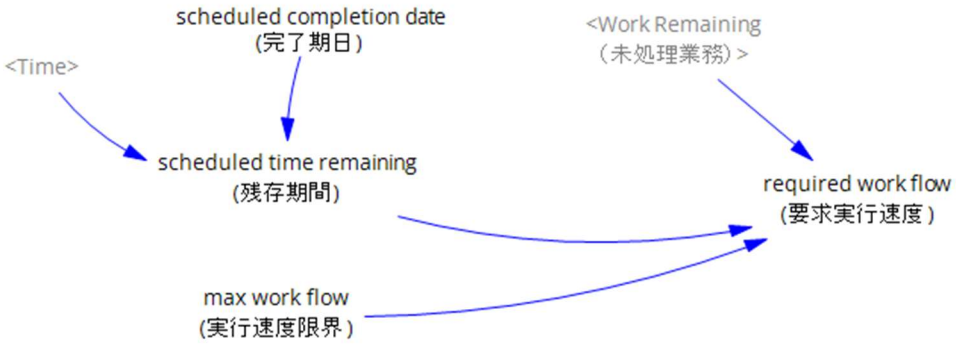


図 *42 project5.mdl* の変更部分（ビュー*2*）

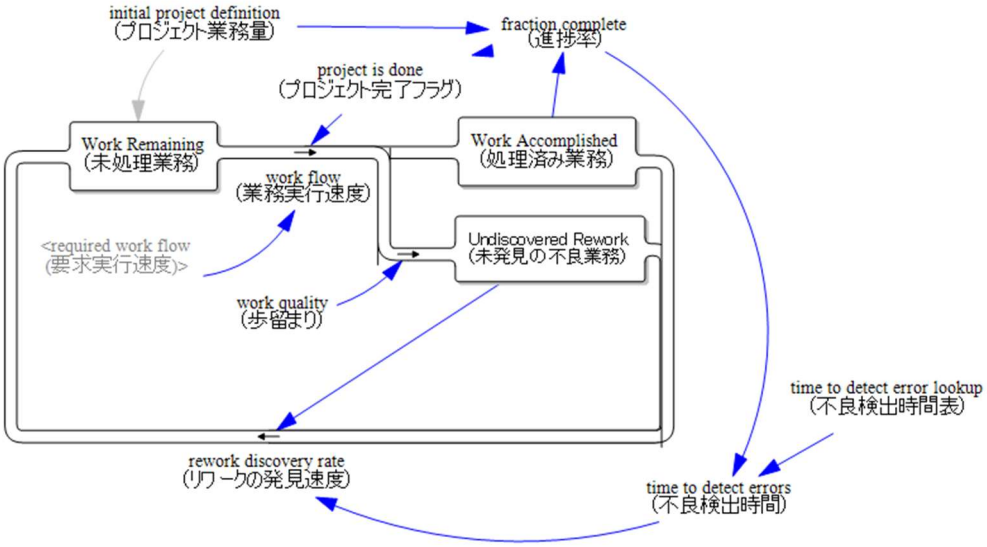


図 *43 project5.mll* の変更なしの部分（ビュー*1*）

Project5.mdl のシミュレーション結果を図 44　project5.mll のシミュレーション結果図 44　project5.mll のシミュレーション結果に示します。

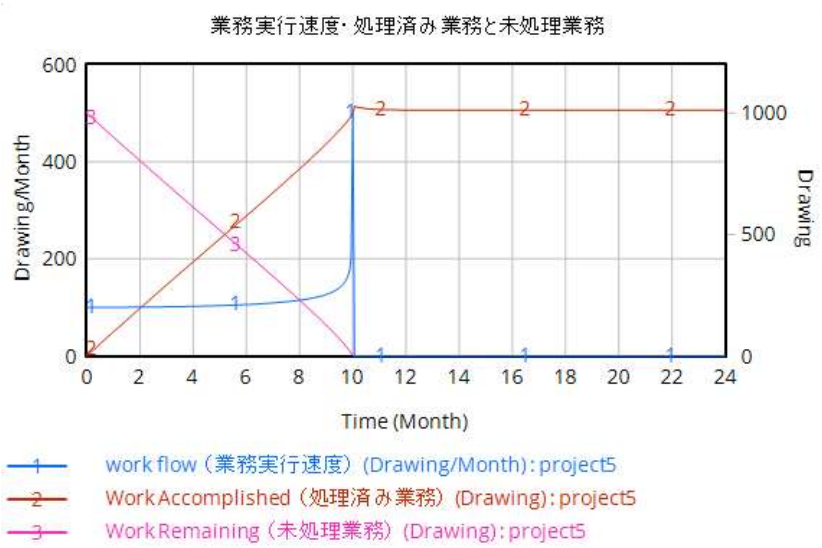


図 *44　project5.mll* のシミュレーション結果

プロジェクトは完了期日通りに完了します。**work flow(業務遂行速度)**はゆっくりと上昇し、次に完了期日通りにプロジェクトを完了するために完了する直前で急激な上昇を示します。

3.6. 労働力と雇用(project6.mdl)

これまでは、仕事を終えるために必要なことは何であれすべて**work flow(業務実行速度)**として扱ってきました。しかし実際に仕事を遂行するためにはリソースが必要です。このモデルでは、**作業レベル(effort)**に注目しリソースは「人」であるとします。これらの人々は、新規に雇用されるか組織内でアサインされるかにかかわらず、仕事を終えるために必要となります。労働力の最もシンプルな定式化は、**労働力－在庫モデル**で使用したものと同じです。この場合、望まれる生産量は、時間通りにプロジェクトを完了するために必要な**work flow(業務実行速度)**ということになります。上記のアイデアをもとに図 42 project5.mdl の変更部分（ビュー2）に対して変更を加えていきます。これらを反映したモデルを図 45　労働力と雇用のモデル化に示します。

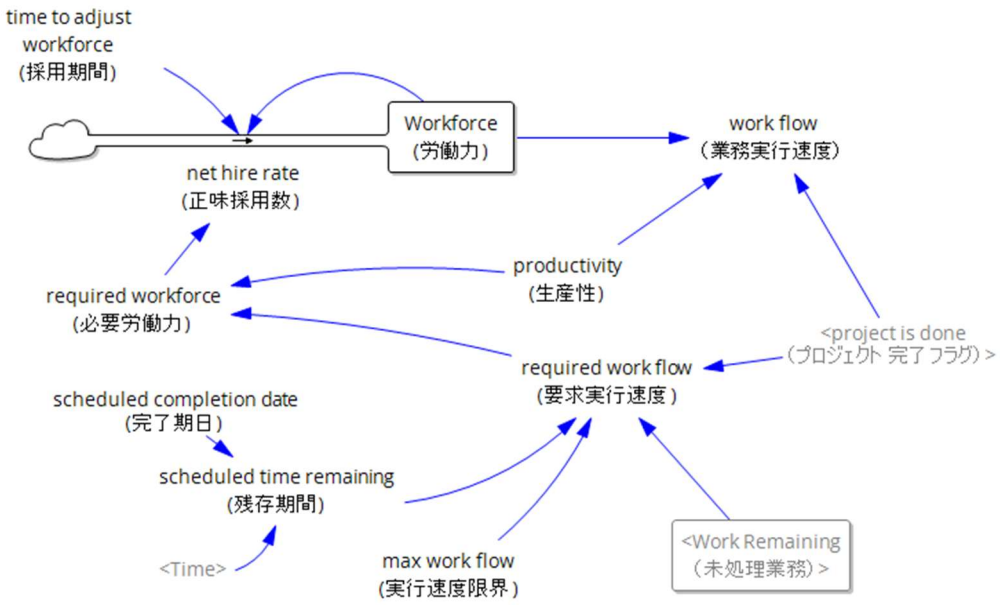


図 *45*　労働力と雇用のモデル化

ワンポイントアドバイス

図 45　労働力と雇用のモデル化を作成するためには、project5.mdl の図 42 project5.mdl の変更部分（ビュー2）からスタートするとよいでしょう。累積的にモデルを発展させるために project5.mdl のファイルを直接修正する場合は、まずファイル名を project6.mdl に変更し project5.mdl を上書きしてしまわないように留意してください。project6.mdl で新たに追加した変数は、変数ツール等を用いてビュー２に追加します。**required workforce(必要労働力)**、**Workforce(労働力)**、**net hire rate(正味採用数)**、**time to adjust workforce(採用期間)**、**productivity(生産性)**がこれに該当します。次にすでに他のビュー（ここではビュー1）で定義されている変数**work flow(業務実行速度)**については、変数ツールではなくモデル変数ツールを用いてこのビュー２に**work flow(業務実行速度)**をスケッチしてください。これはビュー1 の**work flow(業務実行速度)**と実体は同じであることに留意してください。ビュー１、２のいずれの**work flow(業務実行速度)**の式を変更しても、Vensim が自動的に一貫性を保持し、必要により操作

中のビュー以外のスケッチに代行変数等を自動的に追加・削除します。次に、労働力と雇用のモデル化のアイデアに従って、変数間の因果関係の矢印を追加、削除し調整します。

required workflow(要求実行速度)に関する式は、プロジェクトが完了すればゼロになるように変更します。新しい式は次のようになります。

net hire rate = (required workforce - Workforce) / time to adjust workforce
Units: Person/Month
Productivity = 1
Units: Drawing/Person/Month
required work flow = IF THEN ELSE (
project is done,
0,
XIDZ (Work Remaining, scheduled time remaining, max work flow)
)
Units: Drawing/Month
required workforce = required workflow / productivity
Units: Person
time to adjust workforce = 2
Units: Month
Workforce = INTEG (net hire rate, 0)
Units: Person
work flow = IF THEN ELSE (project is done, 0, Workforce*productivity)
Units: Drawing/Month

プロジェクトは誰も作業に携わっていない状態で開始し、作業を進めるために比較的迅速に人員が投入されます。これはプロジェクトの労働力要件とその獲得に関して誰でも考えそうな最も素直な見方です。また計画的な作業強度プロファイルを用いることもあり得ます。これはプロジェクトの当初は緩やかに作業強度を上げ、終了時に向けて徐々に下げていくといったものが考えられます。

以上の修正を反映したモデルをシミュレーションすると、図 46 project6.mdl のシミュレーション結果のようになります。

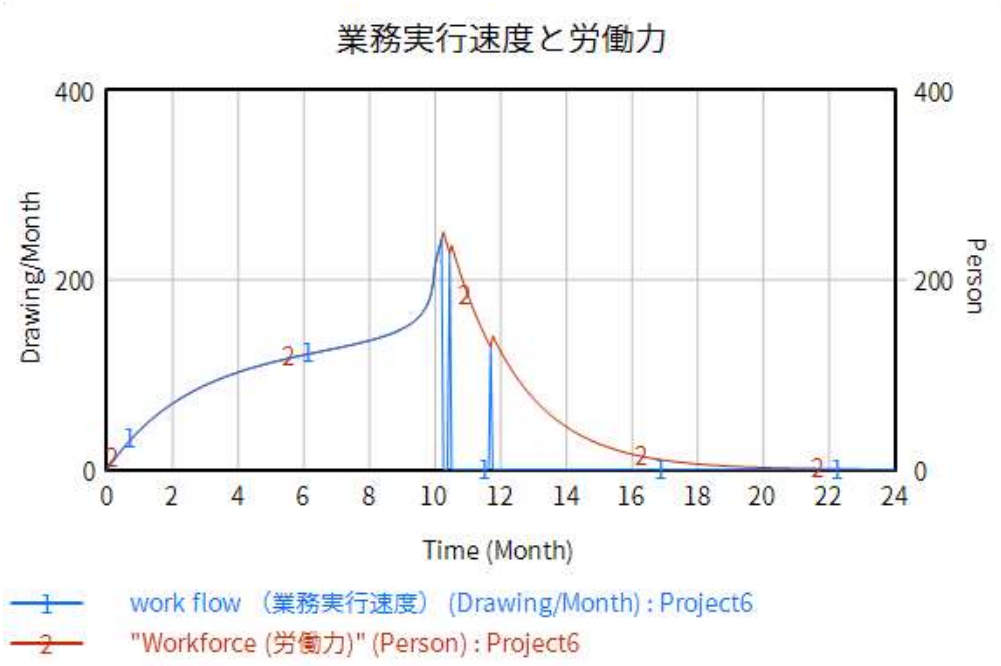


図 *46 project6.mdl のシミュレーション結果*

Workforce(労働者数)は、まず 100 人まで急増し、その後はゆっくりと増え、プロジェクト終了直前になって再び急増します。またプロジェクトが終了した後も活動が再開しています。これらの振る舞いは非現実的ですので両方とも修正が必要です。

3.7. プロジェクトを現実に近づける(project7.mdl)

労働力の調整とプロジェクトの再開の振る舞いを現実に近づけるためにモデルの修正を二つ行います。

3.7.1. 労働力調整意欲

プロジェクトが終盤に近づくにつれて、より多くの人員を投入することは現実的ではありません。なぜならば通常、プロジェクトが進むにつれてチーム内の役割や活動は安定していく傾向があるからです。プロジェクトが 80%完了した後に要員を増やすという判断は多くの場合不適切です。

このような事情をモデルに反映させるため、**willingness to change workforce(労働力調整意欲)**という変数を導入し、不適切なダイナミクスを捉えるようにします。修正後の構造を図 47 雇用調整意欲を導入したモデルに示します。このスケッチを逐次的に作成するために project6.mdl のビュー2からスタートすると良いでしょう。すなわち、変数

ツールで **willingness to change workforce(雇用調整意欲)**と **willing to change workforce lookup(雇用調整意欲表)**を project6.mdl のビュー２に追加します。**fraction complete(進捗率)**はビュー１で定義済みですのでビュー２では代行変数でスケッチし参照できるようにします。

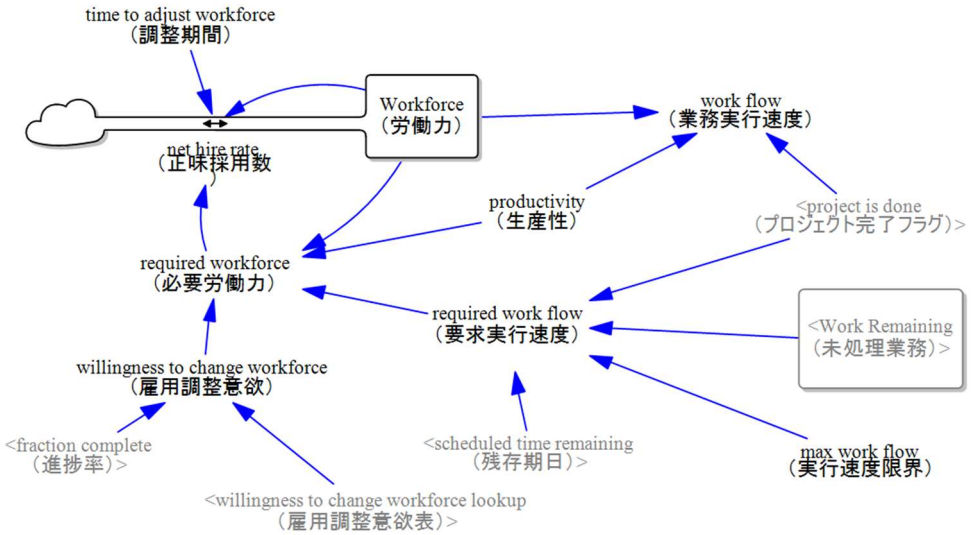


図 *47* 雇用調整意欲を導入したモデル

今回変更した式は次のとおりです。

```
required workforce = IF THEN ELSE (
    Workforce < required work flow / productivity,
    willingness to change workforce * required work flow / productivity
        + (1 - willingness to change workforce) * Workforce,
    required work flow / productivity
)
Units: Dmnl
```

この方程式は、プロジェクトをスケジュールどおり完了させるために必要な労働力が現有労働力より多い場合、すなわち労働力不足の場合に労働力を増強しようとしますが、**willingness to change workforce(雇用調整意欲)**によって増強度合いが調整されます。すなわち **willingness to change workforce(雇用調整意欲)**が高い（１に近い）場合は、**required work flow(要求実行速度)**をより強く反映した **required workforce(必要労働力)**を設定しますが、**willingness to change workforce(雇用調整意欲)**が低い（０に近い）場合は現有労働力である **Workforce(労働力)**のまま維持しようとします。

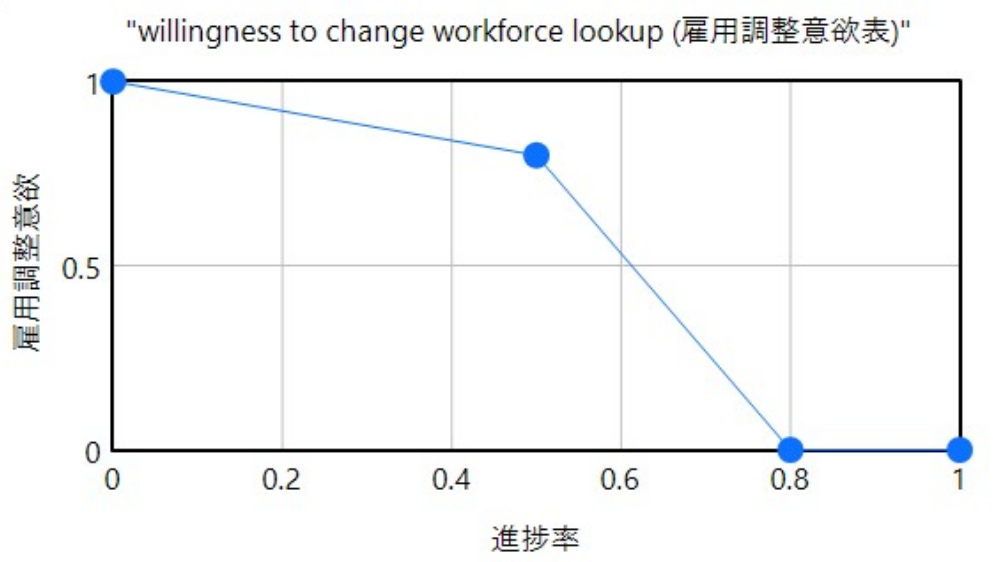
willingness to change workforce =

willingness to change workforce lookup (fraction complete)

Units: Dmnl

willingness to change workforce lookup ((0,1),(0.5,0.8),(0.8,0),(1,0))[5]

Units: Dmnl



willingness to change workforce(雇用調整意欲)は、**fraction complete(進捗率)**によって０から１の値を返します。その関係は表関数 **willingness to change workforce lookup(雇用調整意欲表)**で定義されます。

3.7.2. プロジェクトの再開とモデルの適用

プロジェクトは一度完了したと見なされた後も、問題が発見されて再開することがあります。プロジェクトの再開は頻繁に起こる現象ではなく、非常に大きな問題が発覚した場合に限られます。

ここではプロジェクトを完了させるまでの期間に関心を集中して検討しているため、プロジェクトの再開という問題はさほど重要ではありません。しかし、もし複数のステージを含むプロジェクトにおいて二つ以上の活動が同時に進行し、かつリソースを共有している場合はタスクの再開が重要な問題となり得ます。これをモデル化する方法は、第８章で解説する「サーモスタット問題」の扱い方とほぼ同じです。プロジェクトが一度完了と宣言された場合、よほど状況が悪化しない限り再開はしません。このダイナミクスを**ヒステ**

[5] このモデリングガイドでは、表関数の定義を直感的に理解できるようにするために、表関数にはそのグラフ表示を併記しています。和訳版でもそれを踏襲します。このときグラフは式の一部であると考えて図表番号は付さないこととします。

リシスと呼びます。これらの構造をスケッチするためには、その制御が主たる課題となる**project is done(プロジェクト完了フラグ)**が定義されているビュー１に変更を加えてゆくと良いでしょう。ヒステリシスを実現するための構造は以下の式になります。

project is done = IF THEN ELSE (
project was done :AND: fraction complete > restart fraction,
1,
IF THEN ELSE (fraction complete >= 1, 1, 0)
)
Units: Dmnl
project was done = DELAY_FIXED (project is done, 0, 0)
Units: Dmnl
restart fraction = 0.9
Units: Dmnl

Project7.mdl のビュー１に上記の変更を加えると図 48 ヒステリシスの構造を組み込んだ project7.mdl（ビュー１）に示すようなスケッチになります。

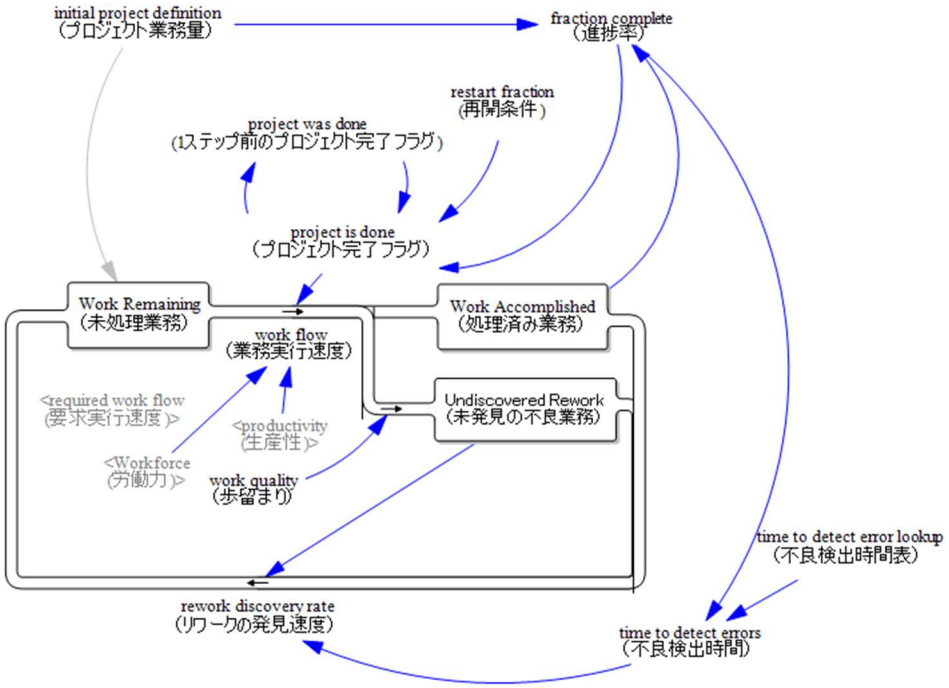


図 *48* ヒステリシスの構造を組み込んだ *project7.mdl*（ビュー１）

3.7.3. 結果として観察される振る舞い

上記二つの修正を加えた構造 project.7mdl を実行した結果を図 49　project7.mdl のシミュレーション結果に示します。この場合、プロジェクトの完了はわずかに遅くなっています。そして活動はプロジェクトの終了時点に向けて緩やかになり平坦になっていきます。プロジェクトが完了した後、わずかですが未発見のリワークが発生しますが、これはプロジェクト再開を引き起こすほどの規模にはなりません。そのため、プロジェクト再開の宣言はなされていません。

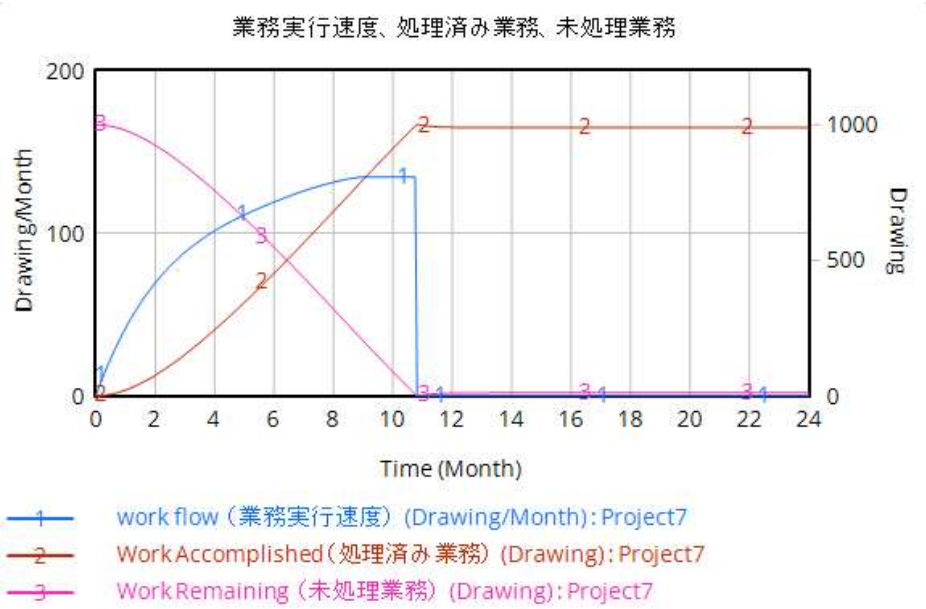


図 *49　project7.mdl* のシミュレーション結果

3.8.　スケジュールによるプレッシャ（project8.mdl）

プロジェクトの終盤に労働力を実効的に凍結することで、スケジュールが活動に与える影響を排除することができました。プロジェクトの最終段階では、人々は通常の速度で淡々と作業を続け、プロジェクトが終了したらすぐに作業を終えることになります。このことから考えると人員レベルは一定であっても、労力の集中度は一定ではないことは容易に想像がつきます。これらを反映した構造を**エラー！ 参照元が見つかりません。** project8.mdl(ビュー３)として示します。

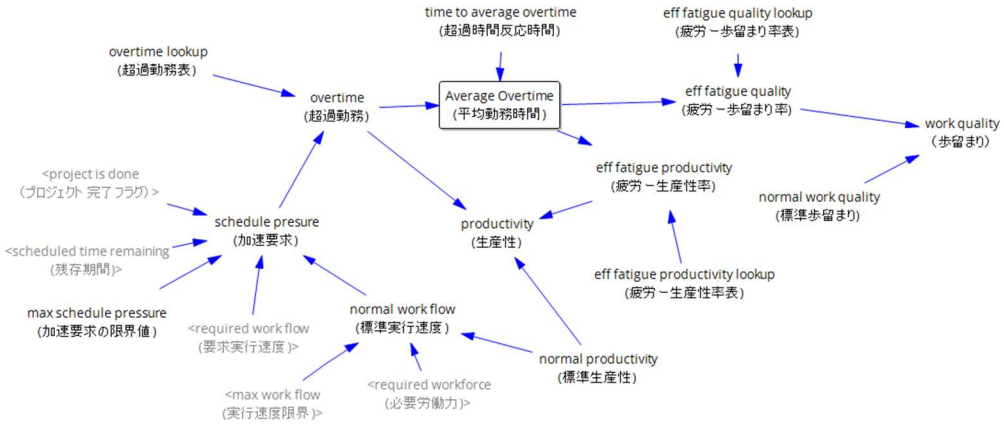


図 *50　スケジュールによるプレッシャを加味したモデル project8.mdl(ビュー3)*

これらのスケッチを描くには project7.mdl ファイルのモデルに新規にビュー３を追加して作成すると良いでしょう。ビュー３の変数のうち、**productivity(生産性)**と **work quality(歩留まり)**はビュー２で定義済なので、モデル変数ツールを用いてビュー３にスケッチします。それ以外の変数はそれぞれの属性あるいは既に他のビューで定義済みなのか否かに従って、ストック変数ツール、変数ツール、代行変数ツールを使用してスケッチします。

以下式について説明します。

まず **overtime(超過勤務)** につながる **schedule pressure(加速要求)**という変数を導入します。そして平均残業時間が大きいと全体的に残業による疲労が蓄積していると考えられるため、**Average Overtime(平均超過時間)**を残業の結果発生する疲労を表す尺度として用います。ここで注意すべきは、**overtime(超過勤務)** や **Average Overtime(平均超過時間)**の発想の起点は超過勤務時間ですが、ここで疲労度の尺度として再定義するため単位は **Dmnl** です。疲労は **productivity(生産性)**と作業の質すなわち **work quality(歩留まり)**を低下させます。式は次のとおりです。

Average Overtime = INTEG (
(overtime - Average Overtime) / time to average overtime,Overtime
)
Units: Dmnl

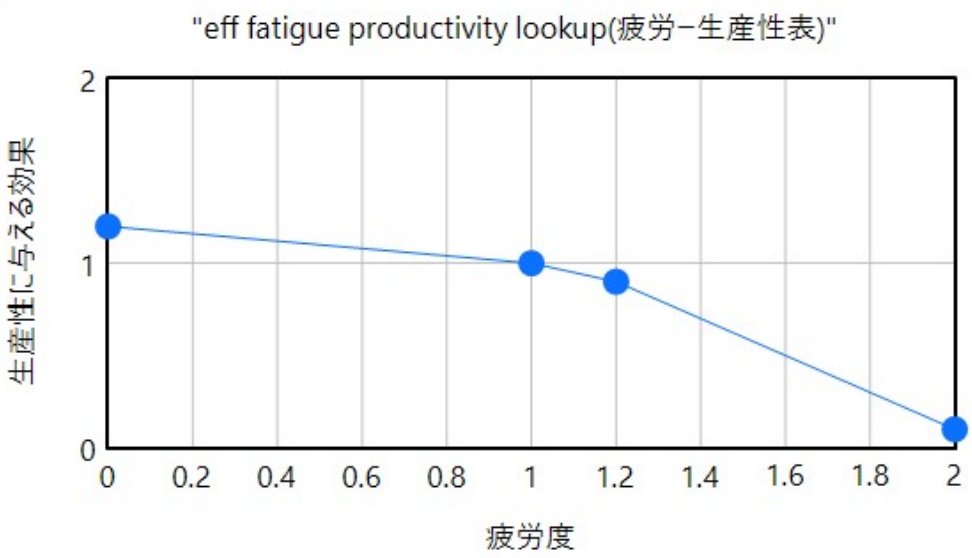
Overtime(平均超過時間)は **overtime(超過勤務)** の**指数移動平均**を表します。直近のデータに最も大きな重みを与え、過去に遡るほどその重みが指数関数的に減少します。こ

のため、**overtime(超過勤務)** の短期的な変動を取り除きつつ長期的な傾向を反映する平均値として機能します。この式は、

Average Overtime = SMOOTH (overtime, time to average overtime)

と同じです。一般的には、**SMOOTH** のような**ダイナミクス関数**は避けるべきです。なぜなら、振る舞いがおかしい場合に、その原因を追究するのが困難になるからです。ダイナミクス関数は、式の見通しが良い場合のみ使用することが重要です。

eff fatigue productivity = eff fatigue productivity lookup (Average Overtime)
Units: Dmnl
eff fatigue productivity lookup ((0,1.2), (1,1), (1.2,0.9), (2,0.1))
Units: Dmnl

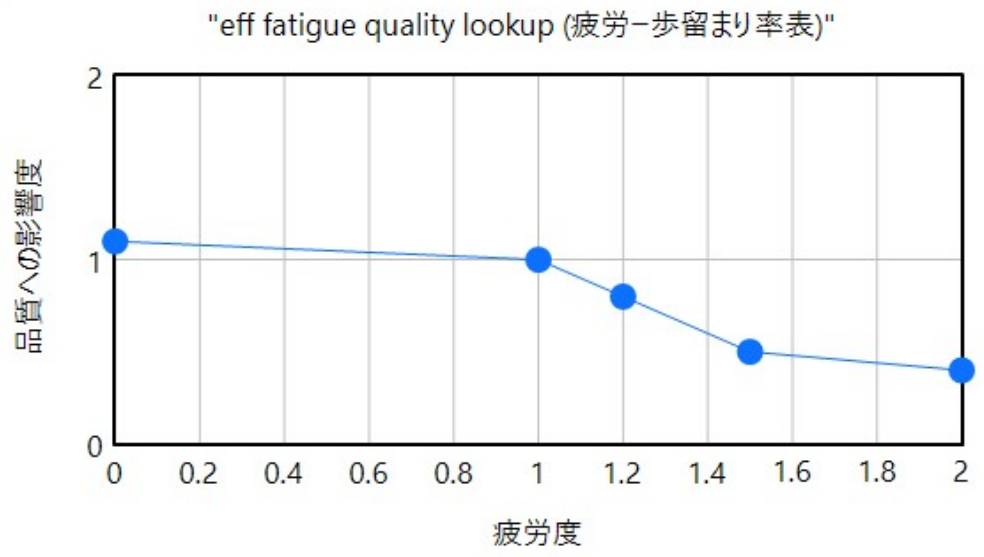


ここで、**eff** は effective の省略形で、**eff fatigue productivity(疲労−生産性率)**は疲労が生産性に与える実効的な影響度合いを意味します。つまり生産性を何らかの方法で認識できるようにしたうえで、疲労が認識された生産性に与えた影響の効果の度合いということになります。例えば作業がプログラミングの場合は、一人のプログラマが一日当たりコンパイルを完了したソースコードの行数への影響度合いといったものになります。

eff fatigue quality = eff fatigue quality lookup (Average Overtime)
Units: Dmnl

eff fatigue quality lookup ((0,1.1), (1,1), (1.2,0.9), (1.5,0.5), (2,0.4))

Units: Dmnl



同様に **eff fatigue quality(疲労−歩留まり率)**は疲労が作業の品質に与える実効的な効果を意味します。**eff fatigue quality(疲労−歩留まり率)**は**Average Overtime(平均超勤時間)**に基づいて、**表関数 eff fatigue quality lookup(疲労−歩留まり率表)**によって定義されます。

max schedule pressure = 5

Units: Dmnl

normal productivity = 1

Units: Drawing/Person/Month

normal work quality = 0.9

Units: Dmnl

normal workflow = MIN (

max work flow,

normal productivity * required workforce

)

Units: Drawing/Month

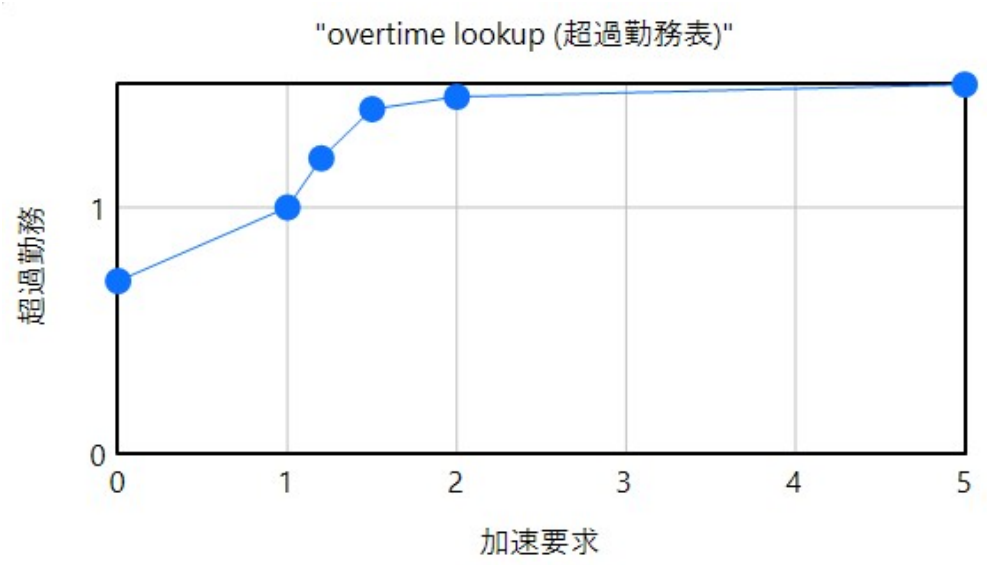
required workforce = IF THEN ELSE (

Workforce < required work flow / normal productivity,

willingness to change workforce*required work flow
/normal productivity
+(1-willingness to change workforce)*Workforce,
required work flow/normal productivit
)
Units: Person

required workforce(要求労働力)はビュー２において定義され、ビュー３において代行変数で参照できるようになっています。**required workforce(要求労働力)**の式の修正では、次に示すように原因変数の変更が必要です。原因変数から結果変数への因果を示す矢印の変更が必要なため、それらの因果関係がスケッチされているビュー２の **required workforce(要求労働力)**に変更を加えます。方程式ツールで式を変更する前に準備として、**productivity(生産性)**から **required workforce(要求労働力)**に至る因果矢印を消去して、**normal productivity(標準生産性)**から **required workforce(要求労働力)**に至る因果矢印を引いておきましょう。**required workforce(要求労働力)**の計算では、**productivity(生産性)**の代わりに **normal productivity(標準生産性)**を使用するように変更します。**normal productivity(標準生産性)**は労働者１人が所定労働時間に疲労度１のときのアウトプット量であり、**productivity(生産性)**は超過勤務も含めた現実の勤務時間をその時点の疲労度で働いたアウトプット量です。もし **normal productivity(標準生産性)**に変更しなければ、超過勤務が考慮された高い生産性で新規人員の投入量が計算されてしまいます。プロジェクトによっては、このようなことは実際には行われてしまっているかしれませんが、そのような場合でも、生産性は直接観察できないため、一定期間あるいは一定のグループにおける決められた方法で導き出された認識された生産性の平均を使用することになります。

overtime=overtime lookup (schedule pressure)
Units: Dmnl
overtime lookup ((0,0.7), (1,1), (1.2,1.2), (1.5,1.4), (2,1.45), (5,1.5))
Units: Dmnl



overtime(超過勤務)は、ここでは疲労度の尺度である**Average Overtime(平均超勤時間)**を決定する要素として用いられる疲労度を表す尺度であり単位は **Dmnl** です。**overtime(超過勤務)**は **overtime lookup(超過勤務表)**に従って、**schedule pressure(加速要求)**から導き出されます。**schedule pressure(加速要求)**に従って**overtime(超過勤務)**が次第に増えていく様子が表関数によって定義されています。

productivity = normal productivity * overtime * eff fatigue productivity
Units: Drawing/Person/Month
schedule pressure = IF THEN ELSE (
scheduled time remaining<= 0 :AND: :NOT: project is done,
max schedule pressure,
ZIDZ (required work flow, normal work flow)
)
Units: Dmnl

スケジュール遅れが生じていない場合は、**normal work flow(標準実行速度)**に対する**required work flow(要求作業速度)**の比率が**schedule pressure(加速要求)**となります。この際に、シミュレーションの開始時には**normal work flow(標準実行速度)**は 0 なので、**ZIDZ** 関数を用います。スケジュール遅れが生じた場合は、**schedule pressure(加速要求)**は完成期日まで、その最大値**max schedule pressure(加速要求の限界値)**となります。

time to average overtime = 2

Units: Month

work quality = normal work quality * eff fatigue quality

Units: Dmnl

これらの変更により、project8.mdl の振る舞いは次のようになり図 51 project8 のシミュレーション結果として示します。

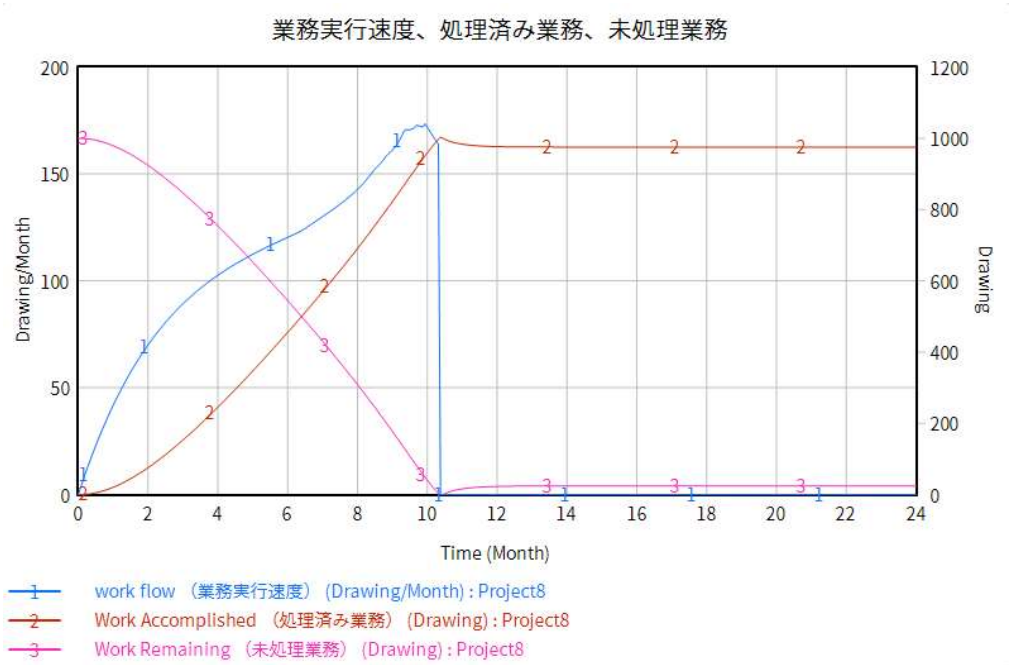


図 *51 project8* のシミュレーション結果

終了に向けた活動のピークは超過勤務で遂行されます。終了付近では作業の質が低下します。また、わずかながら、プロジェクト完了後に発見されるリワークが観察されます。

3.9. **労働力の習熟度別構成**(project9.mdl)

上記のシミュレーションでは、プロジェクト終了後もしばらくの間、人員がプロジェクトに割り当てられたままになっていることがわかります。もちろん、段階的に人員を削減する努力は意味がありますが、現実のプロジェクトの終結に際しては、モデルよりも急激に人員の整理がなされることが多いようです。このことから**人員の獲得と解雇でかかる時間に差がある**と仮定するのが妥当であると考えられます。次に重要な考察点はプロジェク

トに投入する人員の準備です。これまで人員は誰でもすぐに仕事に取りかかれると仮定していましたが、たとえ熟練した人材を投入したとしても**彼らが責任を果たせるスピードに達するまでトレーニングが必要**になります。以上を考慮したモデルを図 52　人材の習熟度を加味したモデルとして示します。

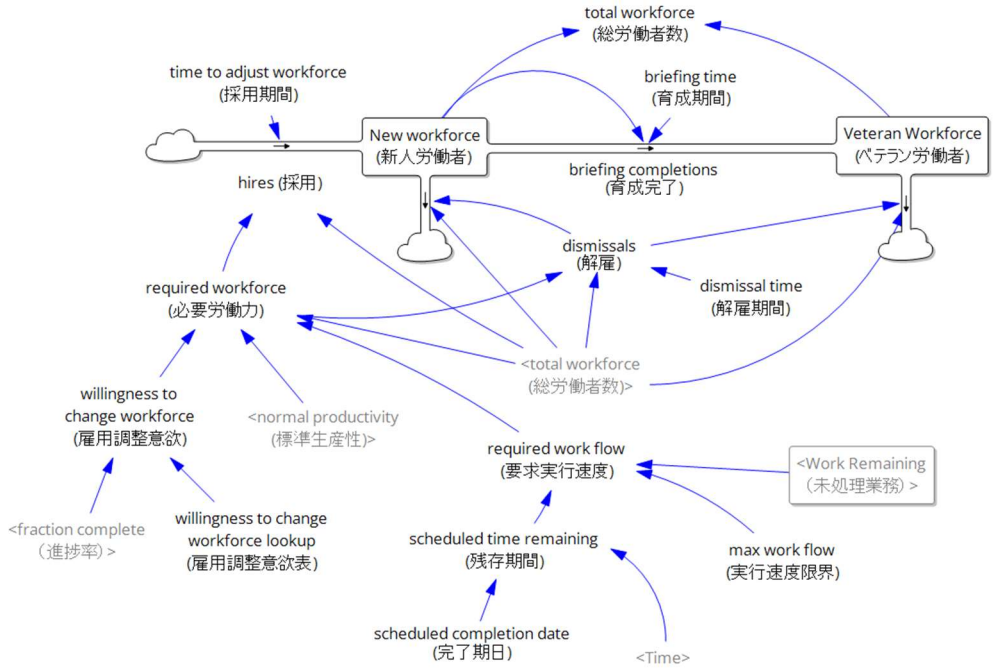


図 52　人材の習熟度を加味したモデル

労働力は２つに分けられます。採用されて入ってきた人員は、まず **New Workforce(新人労働者)**の集合に入ると考えます。その後に十分なトレーニングを経て **Veteran Workforce(ベテラン労働者)**になっていきます。そしてベテランだけが実際に **work flow(業務遂行速度)**に寄与するものとします。したがって **New Workforce(新人労働者)**が多すぎると負担が大きくなってしまいます。一方で人員の必要数を計算する際には、**total workforce(総労働者数)**が用いられます。**hires(採用)**は **new Workforce(新人労働者)**にのみ流入していますが、**dismissals(解雇)**は新規労働者およびベテラン労働者の両方から比例配分して行われるとします。また、解雇にかかる時間は採用にかかる時間より大幅に短く設定されています。

ワンポイントアドバイス

Project8.mdl から累積的に project9.mdl を作成するためには、project8.mdl のビュー２をもとに修正を加えていくのが良いでしょう。Project8.mdl を図 53 project8.mdl のビュー２に示します。

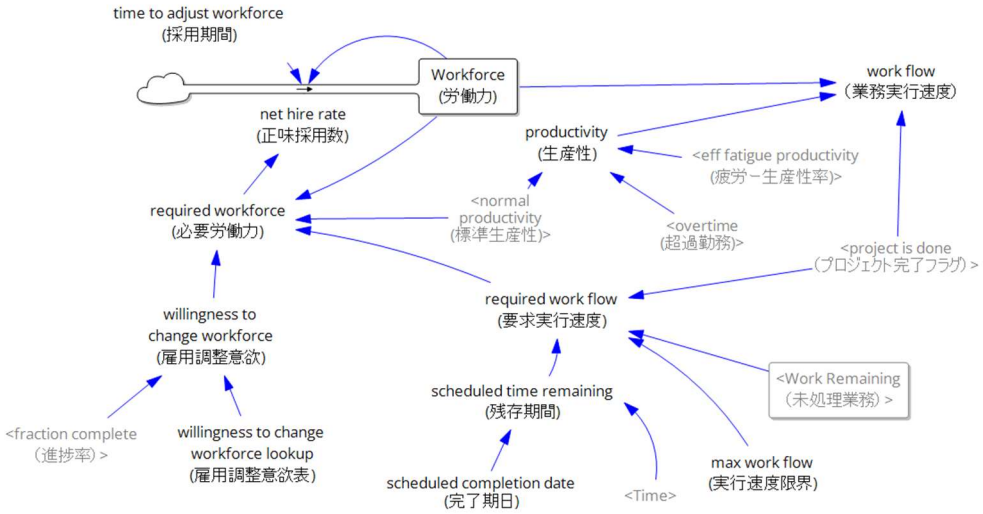


図 *53 project8.mdl* のビュー２

まず Workforce(労働力)を削除して、New Workforce(新人労働者)と Veteran Workforce(ベテラン労働者)のストックに置き換えます。次に net hire rate(正味採用数)は hires(採用)に置き換えます。あとは、その変数の属性や他のビューで定義済みなのか否かで、ストック変数ツール、変数ツール、代行変数ツールを使い分けて必要により変数を置き換えていきます。また、因果関係の矢印も必要により削除・追加・付け替えが必要になります。なお、work flow(業務実行速度)、productivity(生産性)につきましては、work flow の原因変数である Workforce(労働力)を Veteran Workforce(ベテラン労働者)に置き換える必要がありますが、その操作はビュー２で行うよりもビュー１で行った方が因果関係を表す矢印のスケッチが容易です。

変更する式は次のとおりです。

briefing completions = New Workforce / briefing time
Units: Person/Month
briefing time = 2
Units: Month
dismissal time = 0.5
Units: Month

```
dismissals = IF THEN ELSE (
        required workforce < total workforce,
        (total workforce - required workforce) / dismissal time, 0
        )
Units: Person/Month
hires = IF THEN ELSE (
        required workforce > total workforce,
        (required workforce-total workforce) / time to increase workforce,
        0
)
Units: Person/Month
New Workforce = INTEG (
        hires-briefing completions-dismissals
                        * ZIDZ(New Workforce, total workforce),
        0
)
Units: Person
```

ZIDZ は、**total workforce(総労働者数)**が 0 のときに 0 で割ることを防ぎます 。

```
time to increase workforce = 2
Units: Month
total workforce = New Workforce + Veteran Workforce
Units: Person
Veteran Workforce = INTEG (
        briefing completions - dismissals
              * ZIDZ (Veteran Workforce, total workforce),
        0
)
Units: Person
```

Workforce(労働力)を **total workforce(総労働者数)**と置き換えたことにより、**required workforce(必要労働力)**の式も修正が必要です。

required workforce = IF THEN ELSE (

total workforce < required work flow / normal productivity,

willingness to change workforce * required work flow

/ normal productivity

+ (1-willingness to change workforce) * total workforce,

required work flow/normal productivity

)

最後にビュー１で work flow(業務遂行速度)の式を置き換えます。

Work flow = IF THEN ELSE (

project is done, 0, Veteran Workforce * productivity

)

以上の修正を反映したモデルを実行すると、図 54　project9.mdl のシミュレーション結果に示す振る舞いが得られます。

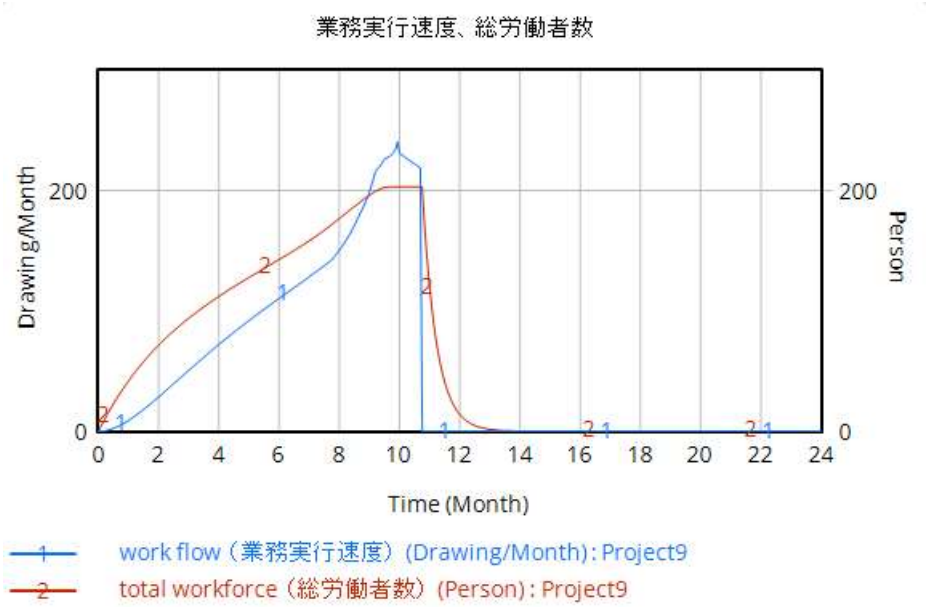


図 *54　project9.mdl* のシミュレーション結果

ここで重要なことは、**total workforce(総労働者数)**から **work flow(業務実行速度)** が逸脱していることと、より多くの人員が瞬間的に必要になっていることです。

ワンポイントアドバイス

Project9.mdl を実行すると、図 55 実行時に表示されるウォーニングメッセージに示すようなウォーニングが表示されます。

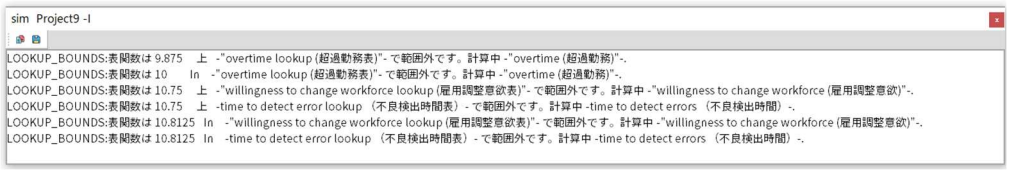


図 *55* 実行時に表示されるウォーニングメッセージ

例えば一番上のメッセージは、時刻 **0.975** において、表関数 **overtime lookup(超過勤務表)** で入力となる **schedule pressure(加速要求)** に、表関数で定義された範囲を超える値が入力されたことを示しています。表関数 overtime lookup(超過勤務表)は、0 から 5 の範囲で定義されていますが、当該時刻に **schedule pressure(加速要求)** は 5 を超えて 10 近辺まで上昇しています。

3.10. 方針の検証(project.mdl)

モデルを少しずつ修正してゆくことで興味深いプロジェクトモデルを開発することができました。このモデルを使って、プロジェクトを上手く進めるための方針を検討してみましょう。検証のために**累積費用**と**プロジェクト実行期間**を知るための式を追加します。その後、雇用に関する方針の選択肢をコンピュータ・シミュレーションし、パフォーマンスの改善が期待できるかを確かめます。

3.10.1. 会計情報を集計するための式

モデル全体を要約するような測定項目を追加することは多くの場合有用です。この例の場合、最も興味深い測定項目は累積費用とプロジェクト実行期間です。それらを測定するための仕掛けを図 56　会計情報を収集するための構造 project.mdl(ビュー4)に示します。Project9.mdl にこの構造を追加する場合は、新規のビュー4 を作成すると良いでしょう。

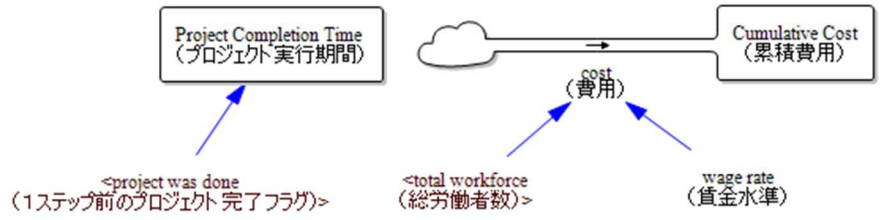


図 *56*　会計情報を収集するための構造 *project.mdl(ビュー4)*

式は次のようになります。

cost = total workforce * labor cost

Units: $/Month

Cumulative Cost = INTEG (cost, 0)

Units: $

Project Completion Time = SAMPLE IF TRUE (project was done = 0, Time, 0)

Units: Month

SAMPLE IF TRUE は、最終的に第一引数の論理式が真から偽に変化した時点の第二引数の値を保持し続けます。つまり **project was done(プロジェクト完了)**が 0、すなわちプロジェクトが終了していない期間は **Time** を返し続け、完了した時点で同じ値を保持し続けます。この定式化により品質の低さが原因で発生する可能性のある再実行を捕捉し、**Project Completion Time(プロジェクト実行期間)**に反映できます。**SAMPLE IF TRUE** は方程式ツールで、タイプをストック、サブタイプを **Sample if True** と指定することで定義できます。

wage rate = 6000

Units: $/Month/Person

wage rate(賃金水準)には利益を含んでいるものとします。

3.10.2. 労働力の上限

労働力に関する方針を検討するための実験として、労働力に上限を設けることを考えてみます。仮に完了すべき図面が 1,000 枚あり、生産性が 1 とすれば、単純な計算で 100 人が 10 ヶ月かけて仕事を終えることができることが分かります。もし人員が 2 倍いたとしたら、おそらく、もっと早くできるに違いありません。必要な労働力あるいは目標とすべき労働力について改めて考えてみることにします。しかし現在の方程式は既に複雑すぎるため、分かりやすくするために **required workforce（必要労働力）** を **indicated workforce（目標労働力）**と名前を変えることにします。その上で改めて **required workforce(必要労働力)**を新しい変数として追加します。新しい構造を図 57 新しい required workforce(必要労働力)の導入に示します。

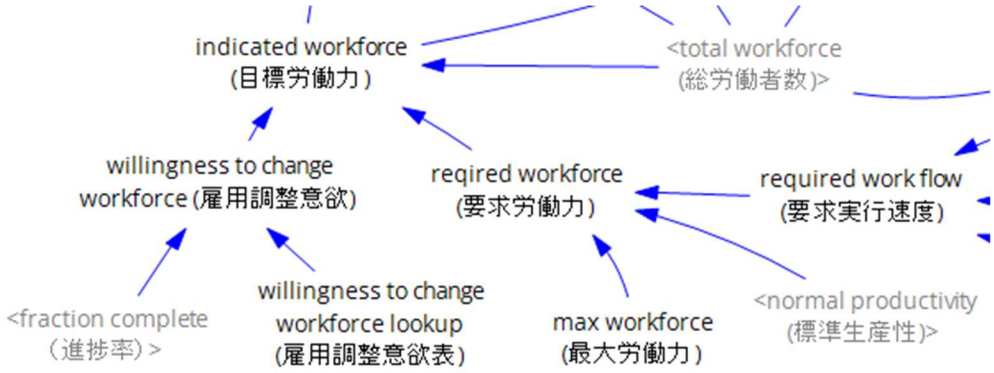


図 *57* 新しい*required workforce(*必要労働力*)*の導入

indicated workforce(目標労働力)は、これまで**required workforce(必要労働力)**が使用されていた場所で使用されることになります。そして、**required workforce(必要労働力)**に対して別途計算方法を定義します。project.mdl のビュー2 において**required workforce(必要労働力)**の名前を**indicated workforce(目標労働力)**に変更したとき、Vensim はそれを参照していたすべての式の中の変数名も自動的に変更します。このようにして、新しい式は次のようになります。以下に旧**required workforce(必要労働力)**と新**indicated workforce(目標労働力)**を対照的に示します。これまでは、必要な労働力を式の中で**required work flow(要求実行速度)**と**normal productivity(標準生産性)**にもとづいて計算により決めていました（式中の下線を付した部分）。その下線部分を新**required workforce(必要労働力)**に置き換えたことになります。

（旧）required workforce(必要労働力)の定義式

required workforce = IF THEN ELSE (
    total workforce<required work flow/normal productivity,
    willingness to change workforce*required work flow/normal productivity
        +(1-willingness to change workforce) *total workforce,
    required work flow/normal productivity
)

（新）indicated workforce(必要労働力)の定義式

indicated workforce = IF THEN ELSE (total workforce <
required workforce,
willingness to change workforce*required workforce +

(1 - willingness to change workforce) * total workforce,

required workforce)

Units: Person

そのうえで、required workforce(要求労働力)を次のように定義します。

max workforce = 1000

Units: People

required workforce = MIN (

max workforce,

required workflow/normal productivity

)

Units: Person

これらの式の変更によって、これまでと基本的に同じメカニズムで **required workforce(要求労働力)**が決定されますが、労働力に制限を加えるためのメカニズムが追加され、パラメータ **max workforce(最大労働力)**で指定することができます。ここでは労働力に関しては無制限とするという方針を試すことを想定して、**max workforce(最大労働力)**は初期値として非常に大きな値(**1000**)が設定されています。もしこれを、例えば 2000 に変更してシミュレーションしても、同じ振る舞いが生成されます。

以上の変更を反映したスケッチを図 58 労働力に上限を設けるメカニズムを組み込んだ project.mld(ビュー2)に示します。

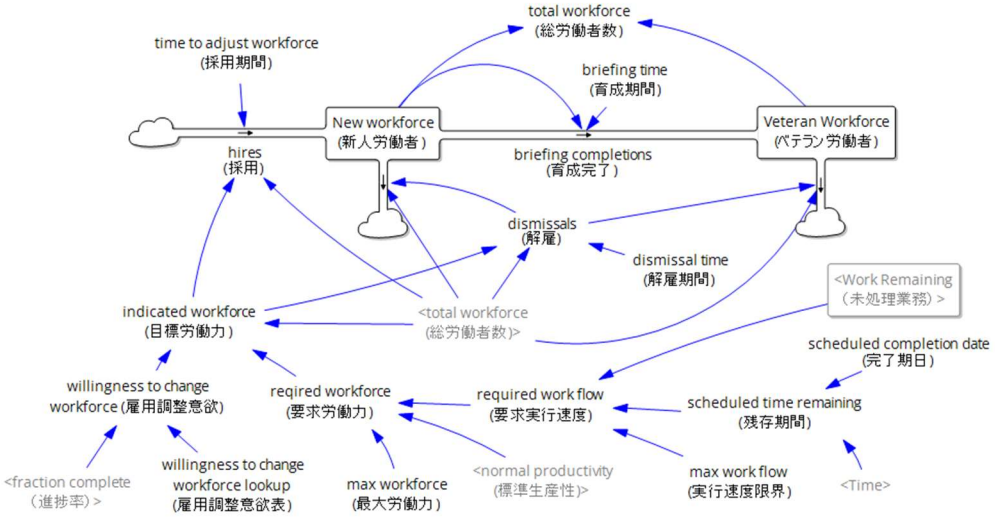


図 *58 労働力に上限を設けるメカニズムを組み込んだ project.mld(ビュー2)*

3. 10. 3. 最終結果の表示

関心を払うべき数字として、累積費用とプロジェクト実行期間という 2 種類の尺度を設定しました。最終的に値がどのようになったかを知るために自作表を使います。「制御パネルの表示」の「自作グラフ」タブの新規自作表を選択します。名称、タイトルを入力後、時間指定設定を 0 から 24 にしたうえで、最終結果のみを表示させたいときは By を 24 とします。変数名は**Cumulative Cost(累積費用)**と**Project Completion Time(プロジェクト実行時間)**を選択します。そして、入出力ツールで自作表を張り付けたうえで実行すると、図 59 表ツールにより表示されたシミュレーション結果のような表が作成されます。

Time (Month)	0	24
"Cumulative Cost (累積費用)" : Project	0	8.85306e+06
"Project Completion Time (プロジェクト実行期間)" : Project	0	10.75

図 *59* 表ツールにより表示されたシミュレーション結果

この表から、累積費用は 8.85 百万ドル、プロジェクト実行期間は 10.75 ヶ月になることがわかります。

3. 10. 4. 労働者数の上限

労働力の上限を 150 人として実験的にシミュレーションを行います。すると、図 60 労働力の上限を 150 人としたとき(**Cap150**)のシミュレーション結果のような結果になります。

図 *60* 労働力の上限を *150* 人としたとき*(Cap150)*のシミュレーション結果

労働力に上限を与えたことで、プロジェクト実行期間は少し伸びます。しかし、コストに関しては少々驚くべき結果が出ています。累積コストは労働力に上限を置いた方が高くなります。なぜでしょうか。**Cumulative Cost(累積コスト)**を選択して、直接原因グラフを使います。制御パネルで時間軸タブを開いてリセットすることを忘れないようにしてください。もし正しい実行結果が出ないときは、データセットタブで正しいデータセットをロードしてください。すると**Cumulative Cost(累積コスト)**に関して直接原因グラフをクリックすれば図 61 Cumulative Cost に関する直接原因グラフのような結果が得られます。

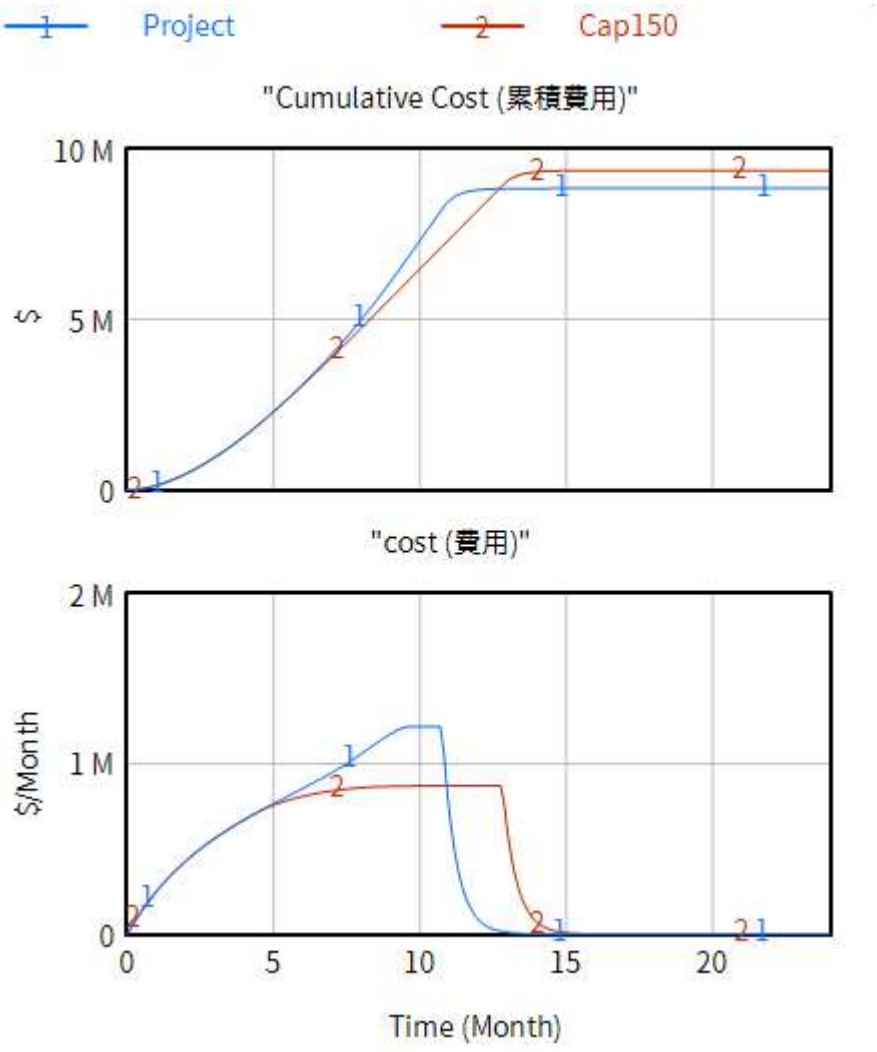


図 *61 Cumulative Cost* に関する直接原因グラフ

上限がない場合(**Project**)に比べて、労働力上限 150 人の **Cap150** は月当たりの人件費は低く抑えられています。しかし、**Cap150** では上限がない場合よりも、より長い期間コストが継続して発生してしまいます。このために累積コストが高くなってしまうのです。次に **productivity(生産性)**を選択して「直接原因グラフ」をクリックしてください。次に **work quality(歩留まり)**を選択して「直接原因グラフ」をクリックしてください。すると図 62 productivity(左図)と work quality(右図)に関する直接原因グラフが表示されます。

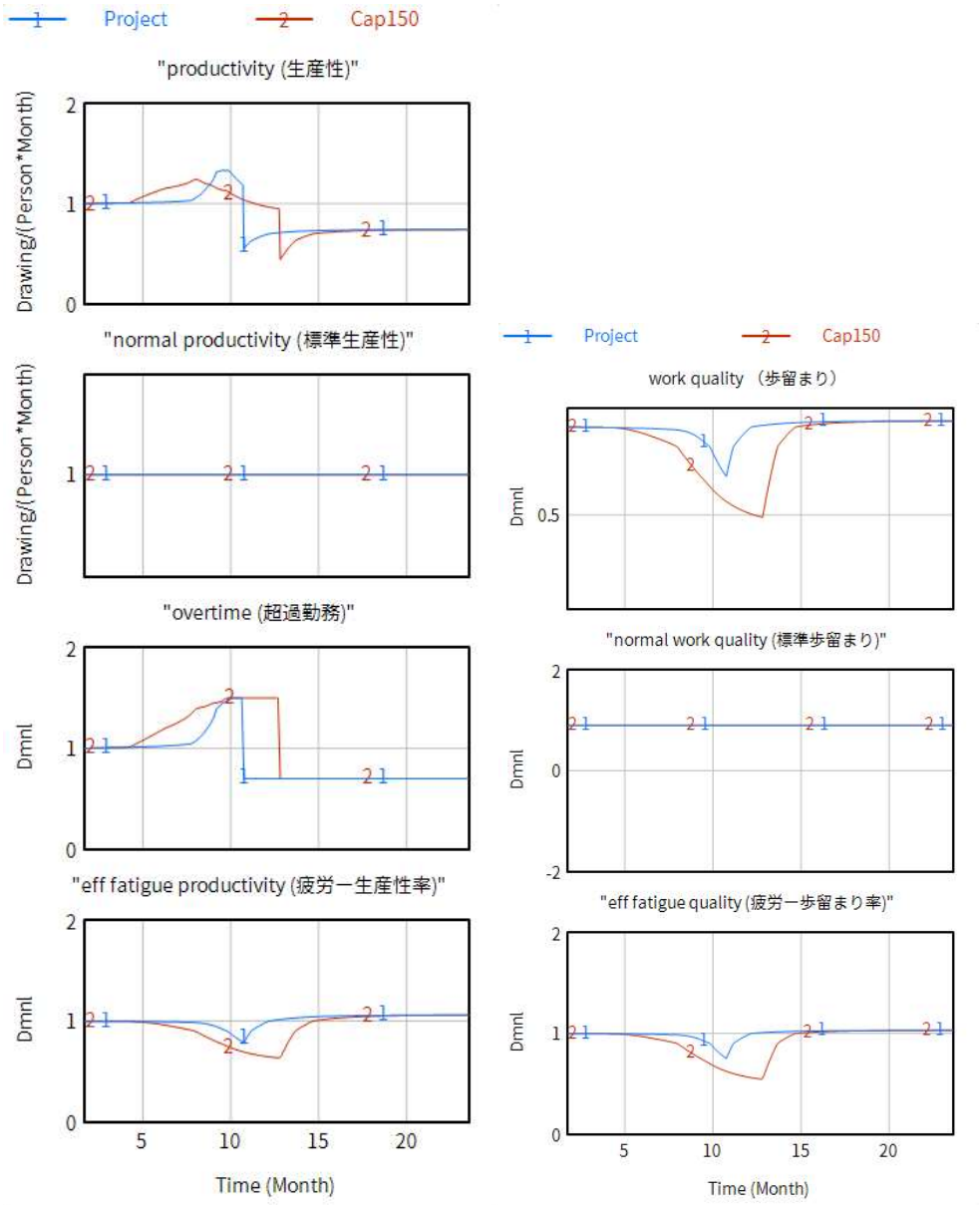


図 *62 productivity(左図)と work quality(右図)に関する直接原因グラフ*

労働力の上限を 150 人とした(**Cap150**)とき、生産性の上昇は限定的になっています。スケジュールからのプレッシャが早い時期からかかり、**productivity(労働力)**を疲弊させ続けています。それに伴って **work quality(歩留まり)**もまた大きく低下していることがわかります。それに対してベースとなるモデル(**Project**)では、労働者が疲労困憊して燃え尽き症候群に陥る前に、アクティビティをどんどん終了させてプロジェクトを完了させてしまうことができるのです。

3.11. 要約

この章では小さな要素を既存のモデルに付け加えていくことで、段階的にモデルを構築していくプロセスを体験しました。この方法の利点は常に動作するモデルがあるという点です。欠点は木を見て森を見失うことにもなりかねないという点です。また、いつ追加を止めるべきかということも難しい問題です。あらかじめ描いた全体像がない場合、理解を深めることなく多くの詳細の追加に突っ走ってしまうことがあります。小さなステップを積み上げることは強力なアプローチですが、その積み上げが望む方向に向かっているかどうかを確かめるために時々自己チェックする必要があります。