目　　次

１．　ブルックスの予言 3
１.1　人月の神話 3
１.2　銀の弾などない 4
1.2.1.　「銀の弾」とは 4
1.2.2.　「銀の弾などない」の論旨 5
１.3　システム開発の本質的作業と付随的作業 6
1.3.1.　創作活動における本質的作業と付随的作業 6
1.3.2.　小説に見る本質的部分と付随的部分 7
1.3.3.　ソフトウェア開発における本質的作業と付随的作業 8
１.4　生産性を大幅に改善できたのは付随的部分 9
1.4.1.　小説書きとワープロ 9
1.4.2.　生産性を劇的に向上させた技術革新 10
１.4.2.1. OS、高水準言語 10
１.4.2.2. UNIX、RBMS、構造化プログラミング 11
1.4.3.　その後の技術革新 12
１.4.3.1. 特効薬が効かなくなった 12
１.4.3.2. 我々は効果の上がりやすいところから手をつけてきた 13
１.5　何故ソフトウェア開発の本質的作業は難しいのか 14
1.5.1.　今ひとつ腑に落ちない点 14
1.5.2.　建築の場合 15
１.5.2.1. 建築の工程 15
１.5.2.2. 建築の本質的作業の進化 16
1.5.3.　ソフトウェア開発の本質的作業の進化を阻むもの 18
１.5.3.1. 不可視性：視覚化しにくい 18
１.5.3.2. 可変性：柔軟すぎる 19
１.5.3.3. 複雑性：量が増大すれば質が複雑化する 19
１.5.3.4. 同調性：環境に依存する 19
１.6　ブルックスの予言は正しかったか？ 20
1.6.1.　銀の弾などない再発射 20
1.6.2.　ブルックスが予言できなかったこと 21

----------下記は試読版（その１）では読めません----------

２．　万人向けでなくなった特効薬
２.1　オブジェクト指向プログラミング
２.2　プロトタイピング
２.3　漸増的開発
３．　1990 年代以降の激変
３.1　急速な技術変化を促したもの
３.2　マイクロプロセッサ革命
３.3　経済のグローバル化による標準化と低価格化・短納期化圧力
３.4　ポスト産業資本主義とソフトウェア業界
４．　2010 年のシステム開発
３．1990 年代以降の激変
４．2010 年のシステム開発

1.ブルックスの予言

1.1　人月の神話

変化の激しいこの業界にあって世界中で 30 年間読み継がれている古典があります。フレデリック・P・ブルックス，Jr.著「人月の神話」です。

ブルックスは、IBM OS/360 用の OS 開発マネジャーを務め、その経験をもとに 1974 年に「人月の神話」を著しました。そこには、システム開発における問題点と解決の指針が記されています。30 年前に書かれた本であるにもかかわらす、今読んでも新鮮で、現在でも多くの本に引用されています。

例えば Google などの検索エンジンで「人月の神話」で検索すると 2000 件以上ヒットします。そして、それぞれのページを見てみると、「人月の神話」が現在でもプロジェクトマネジャーや SE に読まれ、支持され、研究されていることが分かります。

1.2 **銀の弾などない**

ブルックスは 1986 年に「銀の弾などない－ソフトウェアエンジニアリングの本質と偶有的事項」という論文を書きました。これは「今後 10 年間でプログラミング生産性の大規模な改善をもたらすような、ソフトウェアエンジニアリングの発展は一つもないであろう」と予言した論文です。

1.2.1. 「銀の弾」とは

「銀の弾などない」という風変わりな題名の由来は次のとおりです。

> 民話の中の悪夢に登場する怪物のうちでも狼人間ほど恐ろしいものはない。というのも、狼人間は慣れ親しんでいるものを不意に恐怖に変えてしまうからだ。だから、私たちはこの狼人間を魔法のように鎮めることができる銀の弾を捜し求める。
>
> 慣れ親しんだソフトウェアプロジェクトにもこうした性質が若干あり（少なくとも非技術担当マネジャーの目から見ると）、ふだんは無害でまともなのだが、スケジュールの遅延、膨れ上がった予算、そして欠陥製品といった怪物にもなり得る。そして私たちは銀の弾、すなわちコンピュータハードウェアのコストと同じようにソフトウェアのコストも急激に小さくしてくれる特効薬を求める必死の叫び声を聞くのである。
>
> （ブルックス「銀の弾などない」より）

約 20 年前に書かれた文章ですが、今も全然変わっていないと感じるのは私だけでしょうか？

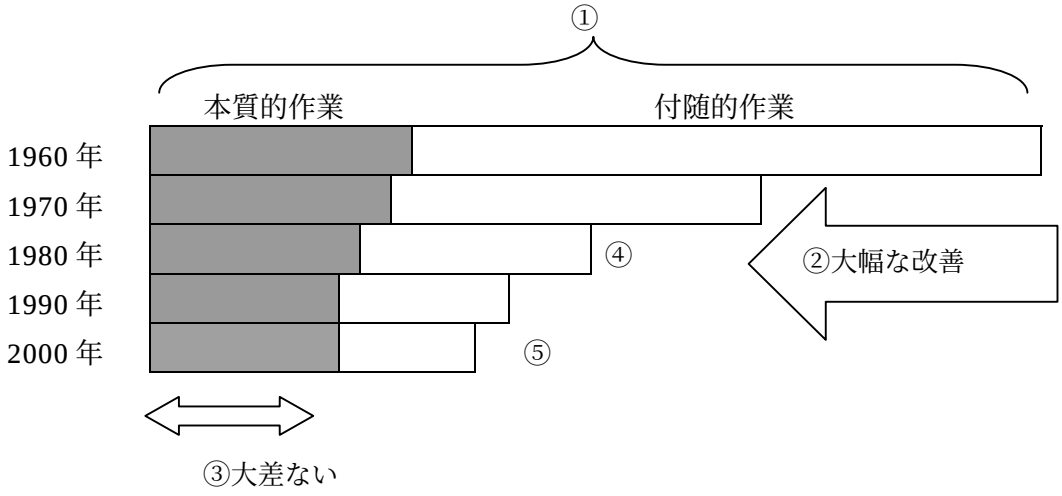
1.2.2.　「銀の弾などない」の論旨

先の引用は次のように続きます。

> しかし、これから十年間という範囲で眺めると、銀の弾などはどこにも見えない。技術においても、管理手法においても、それだけで十年間に生産性や信頼性と容易性での飛躍的な改善を一つでも約束できるような開発は一つとしてない。（ブルックス「銀の弾などない」より）

ブルックスが上記のように断言する根拠は下記のとおりです。

① システム開発には本質的作業と偶有的作業（注）とがある。
② これまでシステム開発において生産性を大幅に改善できた作業は偶有的作業である。
③ システム開発の本質的作業は、ソフトウェアというものが本質的にもつ難しさ（複雑性、同調性、不可視性、柔軟性）故に生産性を大幅に改善できない。
④ 昔はシステム開発の中で本質的作業が占める割合よりも偶有的作業が占める割合の方が圧倒的に大きかったが、偶有的作業の生産性が大幅に改善されたことにより、「銀の弾などない」執筆時点（1986 年）では本質的作業の割合の方が大きくなっている。
⑤ したがって、今後 10 年間でプログラミング生産性の大規模な改善をもたらすような、ソフトウェアエンジニアリングの発展は一つもないであろう。例えば、付随的作業が全体の 40%であるとすれば、たとえ付随的作業の生産性が 2 倍になったとしても、それによる全体のコストの削減は 20%にすぎない。



（注）本書では「偶有的」という言葉はあまりにも難解なので、これ以降、引用部分以外は「付随的」という言葉に置き換えます。

1.3 システム開発の本質的作業と付随的作業

1.3.1. 創作活動における本質的作業と付随的作業

それでは、ブルックスは何をシステム開発の本質的部分と考え、何を付随的部分と考えたのでしょうか？

ソフトウェア開発は、彫刻、工芸、建築、映画、小説、作曲などと同じく創作活動です。そして、全ての創作活動は表現媒体を必要とします。彫刻家にとっては石や木、ピアニストにとってはピアノが表現媒体です。
したがって、創作活動には必然的に次の二つの作業が含まれるのです。
① 何を作るかを創作者が頭の中で構想する作業。
② 構想を表現媒体に実装する作業。

彫刻はまず彫刻家の頭の中でイメージされ、次にそのイメージが石や木に表現されます。
①が創作活動の本質的部分であり、②が付随的作業です。そして、本質的作業と付随的作業にはそれぞれ異質の難しさが存在します。

高村光太郎作　蝉（大正 13 年）

高村光太郎の彫刻刀

１.3.2.　小説に見る本質的部分と付随的部分

創作活動における本質的作業と付随的作業について、小説を例にして、もう少し考えてみましょう。

小説を書くという作業は下記の工程を踏みます。

① 粗筋作り	② 文章書き/推敲(手)	③ 版下作成	④ 校正	印刷

【本質的作業】

① 小説家はまず「粗筋作り」をします。これは登場人物の設定や効果的な構成、骨組みなど小説の核となるアイデアを考える作業です。この中には各種文献を調べたり、現地取材をしたりする作業も含まれます。例えば戦国時代の堺を舞台とした歴史小説を書く場合、小説家は膨大な史料を読むだけではなく、実際に堺に行き、船に乗り海上から堺を眺め、400 年前の人々が見た風景を想像し、イマジネーションを膨らませるかもしれません。小説家はこうした作業で材料が出揃ってから文章を書き出します。

【付随的作業】

② 小説家は上記構想を原稿用紙に書き、何度も推敲を重ね、何度も書き直します。推敲の過程で粗筋にも変更が入ることもあります。つまり、①の粗筋作りの工程をやり直すこともあるのです。

③ 原稿が完成したら印刷屋に版下の作成を頼みます。

④ 版下が出来上がってきたら校正します。但しこの段階で許されることは誤字脱字のレベルの小さな修正のみです。その後は印刷の工程に入ります。

ここでは①の「粗筋作り」が小説を書くという作業の本質的な作業で、②以降の作業が媒体に表現する付随的作業です。②以降の作業も重要な作業ですが、小説の価値を決定づけるものは①の「粗筋作り」だからです。

１.3.3.　ソフトウェア開発における本質的作業と付随的作業

すべてのソフトウェア構築には、本質的作業としての抽象的なソフトウェア実体を構成する複雑な概念構造体を作り上げること、および偶有的作業としてそうした抽象的実在をプログラミング言語で表現し、それをメモリスペースとスピードの制約内で機械言語に写像することが含まれている。

（ブルックス「銀の弾などない」より）

フレデリック・P・ブルックス

前節では、小説書きの本質的作業は粗筋作りであると述べました。
ソフトウェア開発の本質的作業は、ソフトウェアの粗筋作り、すなわち、ソフトウェアのコンセプトを作り、外部仕様を決め、内部構造を設計し、アルゴリズムを考えることです。
ブルックスはこのことを「抽象的なソフトウェア実体を構成する複雑な概念構造体を作り上げること」と表現をしています。

また、小説書きでの付随的作業とは「文章書き、推敲、版下作成、校正」でした。
ソフトウェア開発でこれに相当するものは、「プログラムを書き、単体でテストし、修正し、他の部分と結合し、テストし、修正する」という作業です。これをブルックスは「抽象的実在をプログラミング言語で表現し、それをメモリスペースとスピードの制約内で機械言語に写像すること」と表現しています。これがソフトウェア開発における媒体に表現する作業であり、付随的作業です。
詳細設計が本質的作業なのか付随的作業なのかは、その詳細設計の中身によります。要するに「ソフトウェアの粗筋作りだ」と言えるレベルの詳細設計は本質的作業であり、「プログラムを書く直前の補助的作業」言えるレベルの詳細設計は付随的作業です。

1.4 **生産性を大幅に改善できたのは付随的部分**

1.4.1. 小説書きとワープロ

次に小説書きとワープロを例にして、技術革新が創作活動に与えた影響を考えてみましょう。

前節で小説書きの工程を下記のように説明しました。

① 粗筋作り・・・・・・・・・・・・本質的作業
② 文章書き、推敲
③ 版下作成　　・・・・・付随的作業
④ 校正

【ワープロ登場前】

① 粗筋作り	② 文章書き/推敲(手)	③ 版下作成	④ 校正	印刷

【ワープロ登場後】

① 粗筋作り	②③④ 文章書き/推敲 (PC)	印刷

ワープロは②～④の作業を劇的に変えました。
②の修正作業を飛躍的に容易にしたのです。手書きでは数時間かかった修正が瞬時に行えるようになりました。また、③版下作成と④校正の作業がなくなりました。②の作業の中に③④の作業が含まれてしまったのです。
それに対し、本質的作業に要する時間はワープロがあろうと無かろうとほとんど変わりません。データの整理などで若干の貢献があるのかもしれませんが・・・。
松本清張も司馬遼太郎も万年筆で原稿を書いていました。彼らがワープロを持っていたらもっと傑作が書けたということはあり得ないのです。
つまり、ワープロという技術革新は本質的作業にではなく付随的作業に集中的に作用したのです。

１.4.2.　生産性を劇的に向上させた技術革新

前節では、小説書きにおいてワープロという技術革新は本質的作業にではなく付随的作業に集中的に作用したと述べました。
ソフトウェア開発の歴史を振り返ってみると、ソフトウェア開発での数々の技術革新も本質的作業ではなく付随的作業に集中的に作用したことが分かります。

ブルックスも「銀の弾などない」の中で「過去におけるソフトウェア生産性における大きな収穫のほとんどは、偶有的作業をはなはだ困難にする人為的障害を除去することによって得られた」と言っています。この部分について考察してみましょう。

１.4.2.1. OS、高水準言語

50 年に及ぶソフトウェア開発の歴史の中で、開発生産性を劇的に向上させた技術革新がいくつかありました。その最大のものは、OS の発明と高水準言語の発明です。

OS が発明される前、プログラマはハードウェアの制御を意識しなければならず、それがプログラマの構想を実現する上での大きな障害となっていました。OS はプログラマをハードウェアの制御という重荷から解放しました。OS は正しくコンピュータという媒体に表現することの難しさを除去する発明でした。つまり付随的作業の難しさを劇的に軽減する特効薬でした。

また、高級言語が発明される前、プログラマはビット、レジスタ、スタックなどの抽象度の低い概念を使って、抽象度の高い構想を実現しなければなりませんでした。高級言語は「変数」「配列」「サブルーチン」「関数」などの扱いやすい概念を提供しました。これによって、プログラムミング作業は通常の文章を書く作業に近づきました。高級言語もまた媒体に表現することを容易にする特効薬でした。

OS の発明と高水準言語の発明によって、システム開発の生産性は 10 倍位向上したでしょう。

１.4.2.2. UNIX、RBMS、構造化プログラミング

OS と高水準言語の次に重要な技術革新として UNIX、RBMS、構造化プログラミングがあげられます。

UNIX はツリー構造のファイルシステム、標準入出力などの高度な抽象化によって、プログラミングを容易にしました。それだけでなく、使い勝手のよい開発環境を提供することによって、ソフトウェア開発を容易にしました。UNIX が登場するまでは、プログラマはコンパイルすら自由にできず、コンパイル回数を減らすためのシンタックスレベルの机上デバッグに多くの時間を費やしていました。UNIX の登場は、ワープロが小説家に修正が容易で且つその効果を即座に評価できる環境をもたらしたことと似ています。UNIX もまた媒体に表現することを容易にする発明でした。

RDBMS は非常に分かりやすく扱いやすいデータベースモデルを提供しました。また、構造化プログラミングは制御構造を制約することによって、プログラムがスパゲッティ化することを防ぎ、バグの発生を減少させるとともに、プログラムの保守性を向上させました。RDBMS も構造化プログラミングも媒体に表現することを容易にする発明でした。

これらの技術革新によってソフトウェアの開発生産性は 3 倍～5 倍位向上したと思います。

１.4.3.　その後の技術革新

１.4.3.1. 特効薬が効かなくなった

その後も、オブジェクト指向プログラミング、UML、プロトタイピング、漸増的開発、さらには RUP（RationalUnifiedProcess）や XP（eXtreme Programming）等、多くの技術・技法が現われました。しかし、これらの発明は「1.2.1　生産性を劇的に向上させた技術革新」で例示した発明ほどには劇的な効果をもたらしていません。

例えば、高水準言語がソフトウェアの開発コストを急激に小さくしたことに異を唱える人はいません。あるいは、UNIX で実現されたプログラミング環境がプログラマの生産性を大きく向上させたことを否定する人もいません。
しかし、最近登場した発明は大概、評価が分かれます。「効果が出た」という人がいる一方で、「あまり効果がでなかった」という人が必ずいます。さらに、「むしろ有害だった」という人さえいます。例えば「ウォーターフォールは良かった」と古き良き時代を懐かしむ声は根強いものがありますし、「オブジェクト指向を採用してかえってコストがかかった」という話しもよく聞きます。最近の特効薬の効果は、かつての特効薬の効果ほどには圧倒的なものではありません。誰が飲んでも効くという薬ではないのです。

多くの人々はソフトウェア技術の進歩は年々加速していると思っています。しかし、実は、個々の技術革新がソフトウェア開発生産性に与える影響は時代が下るにつれて小さくなっているのです。

１.4.3.2. 我々は効果の上がりやすいところから手をつけてきた

これまでに劇的な効果を上げてきた発明は、付随的作業の難しさを除去する発明でした。OS や高級言語といった付随的作業の難しさをに関する特効薬は誰が飲んでも一定の効果が出ます。一方、プロタイピングや漸増的開発などの本質的作業に関する特効薬は飲む人によって効果が大きく異なります。それは小説書きにおいて粗筋作りを容易にするための万人向けの技術が存在しないことと似ています。

我々は効果の上がりやすいところから手をつけてきました。それが、比較的初期に発明された特効薬が近年発明された特効薬よりも劇的効果を発揮した理由なのです。

「OS、高級言語、UNIX、RDBMS、構造化プログラミング、オブジェクト指向プログラミング、プロトタイピング、漸増的開発」という順番は、普及の順番でもあるし、効果が劇的であった順番であるし、攻略対象が付随的難しさから本質的難しさへと向かう順番でもあるのです。

オブジェクト指向プログラミング、プロトタイピング、漸増的開発については「2. 万人向けでなくなった特効薬」で詳しく解説しています。

1.5 何故ソフトウェア開発の本質的作業は難しいのか

1.5.1. 今ひとつ腑に落ちない点

前節では、ソフトウェア開発 50 年の歴史を振り返り、本質的作業を対象とした発明が劇的効果をもたらしてこなかったことを明らかにしました。

また、小説の粗筋作りがどの時代にあっても難しいこと、そしてそれが技術の進歩によって生産性が改善されるという性質のものではないことは容易に理解できます。

しかし、ソフトウェア開発の本質的作業がどの時代にあっても難しいということについては、今ひとつ腑に落ちない点があります。
システム開発は、作家が自らの意志とアイデアをもって一人で書き上げる小説と違い、大規模な請負事業です。小説よりもはるかに経済性、社会性、事業性の高い活動です。それ故、これまでの数十年間、全世界の企業、学者、技術者がシステム開発の本質的作業を効率化しようとして莫大な資本と優秀な頭脳を投入してきました。それでも攻略できない難しさの正体とは一体何なのでしょうか？

1.5.2. 建築の場合

ソフトウェア開発における本質的作業の難しさの正体を明らかにする前に、本質的作業の難しさが軽減されるということがどのようなことなのか、建築を例にして見てみましょう。

1.5.2.1. 建築の工程

たとえば、プレハブ住宅の建築会社に家作りを頼んだら、下記のような工程で家が出来上がります。

①	設計・見積・契約	建築会社が顧客から要望を聞き、それに基づくプランと資金計画を顧客に提案します。顧客が納得するまで何度もそれを修正します。
		建築会社が詳細図面と見積書を顧客に提出します。顧客が納得するまで何度もそれを修正し、顧客が納得すれば請負契約が成立します。
②	製造	建築会社は設計に基づいて、工場で家の部品を作成します。プレハブ住宅の場合、家の大部分を工場で作ってしまいます。
		現場で基礎工事をします。
		工場から家が運ばれてきて、設置されます。
		内装は現地で行います。
③	テスト・引渡し	建築会社が施工管理チェックします。
		顧客立会いのもと細部をチェックします。
		建築会社は完成した家を顧客に引渡します。

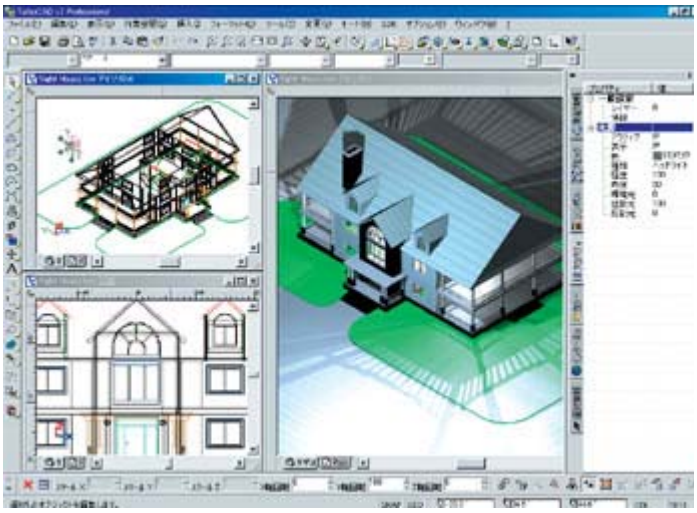
「ソフトウェア開発ではどうしてこのように整然と仕事が進まないのだろう」というソフトウェア技術者の嘆息が聞こえてきそうです。

１.5.2.2. 建築の本質的作業の進化

(A)概観

建築の本質的作業は「①設計・見積・契約」の作業です。この部分の難しさが、IT 技術の進歩によって次のように大幅に軽減されました。

(1) CAD ソフト及び建築ソフトにより、基本設計から詳細図面の作成にいたるまでの建築設計全体が効率化され、品質も高まりました。速くなっただけでなく、正確で精緻な図面が書けるようになったのです。
(2) 構造計算、積算計算、見積計算が各種ソフトウェアによって速く且つ正確に行えるようになりました。
(3) CAD ソフト及び CG ソフトにより正確で表現力豊かなプレゼンテーションを敏速に行えるようになりました。これにより顧客が完成像を正確にイメージできるようになりました。



TURBOCAD v8 の画面

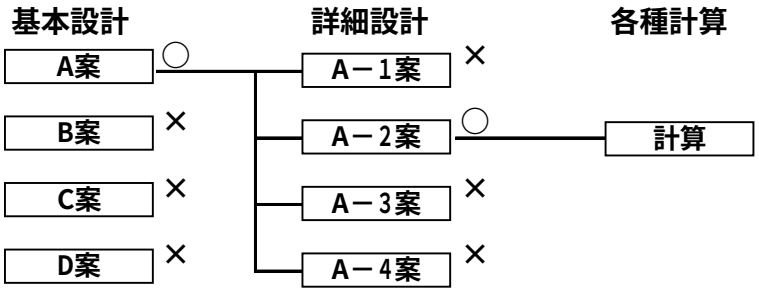
上記のとおり、設計、計算、プレゼンテーションの質と速度が劇的に向上した結果として、建築の本質的作業は質的に進化しました。
設計段階で様々な代替案を出せるようになったのです。しかも、設計・計算・プレゼンテーションのセットの代替案を。
もちろん、それまでも基本設計、詳細図面、各種計算、プレゼンテーションなどの個々作業の中では様々な代替案が出されました。しかし個々の作業に時間と労力がかかったため、セットの代替案を複数作成することは不可能だったのです。

(B)**従来の建築の本質的作業**

下図は従来の建築の本質的作業を表しています。

基本設計の段階で様々な案が検討されます。しかし、詳細図面を作成することに時間と労力がかかったため、詳細設計は採用された基本設計についてのみ行われました。詳細設計でも複数の案が出されます。しかし、各種計算に時間と労力がかかったため、各種計算は採用された詳細図面についてのみ行われます。計算の結果、問題が発生すれば、採用された詳細図面を若干修正します。

顧客に対する分かりやすいプレゼンテーションはありませんでした。

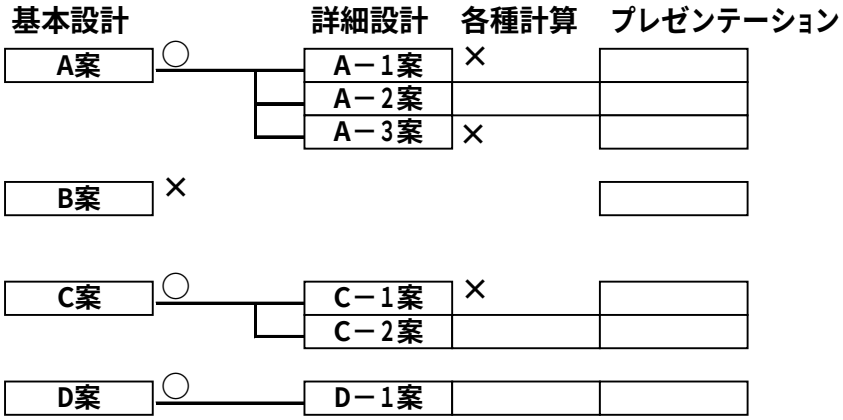


(C)**進化した建築の本質的作業**

下図は進化した建築の本質的作業を表しています。

詳細図面の作成、各種計算、プレゼンテーションにかかる時間とコストが激減したので、複数の基本設計について複数の詳細図面が作成し、複数の詳細図面について各種計算やプレゼンテーションを行うことが可能となりました。

顧客と建築デザイナーは設計段階で以前よりも多くのパターンやオプションを試せるようになったのです。



1.5.3. ソフトウェア開発の本質的作業の進化を阻むもの

建築では本質的作業が進化を遂げたのに、何故ソフトウェア開発では本質的作業が進化しないのでしょうか？

建築での本質的作業の進化の前提として、建築が持つ下記の性格があります。

- 設計段階で全工程を細部にいたるまで見通せる。
- 契約前に顧客と建築会社が詳細な完成イメージを共有できる。
- 契約後の仕様変更がない。

実は、ソフトウェア開発はこれと正反対の性格と持っているのです。

1.5.3.1. **不可視性：視覚化しにくい**

建築は 3 次元空間に物理的に存在するものです。したがって設計段階でも本質的に視覚化しやすく、素人でも設計図を見てどのような建物ができるのか十分に理解できます。それ故に契約前に顧客と建築会社が詳細な完成イメージを共有できるのです。また、建築デザイナーと工場、建築現場との間でも誤解が生じません。

それに対して、ソフトウェアは人間の思考という形のないものから生まれ、最終形はビットのオン・オフとも、不特定多数の人々の使用とも言える創作物です。ソフトウェアはどの段階でも自然には視覚化できないものです。

それ故にソフトウェア開発では、契約前に顧客と建築会社が詳細な完成イメージを共有できません。また、設計者と実装者との間で詳細な完成イメージを共有することも難しいのです。

例えば楽譜を見ても素人はどのような曲であるか理解できません。演奏されて初めて分かるのです。玄人なら分かるかというと、同じ楽譜でも解釈に大きな幅があります。ソフトウェアは視覚化しにくいという一点では音楽に似ています。しかしながら、音楽と違って極端な厳密性が要求されます。

１.5.3.2. 可変性：柔軟すぎる

建築の場合は、途中での仕様変更がいかに大変なことか、素人でも直感的に理解できます。一方、プログラムは極端に自由で可塑的な素材なので、顧客もプログラマも仕様変更を安易に考えてしまいます。
また、視覚化のしにくさ故に設計段階では完成像をイメージできないため、工程が半分以上進み、おぼろげながら完成像が見えてきた段階で、顧客は「自分のイメージと違う」と言い出します。
つまり、システム開発での仕様変更（顧客の側から見たら変更ではない）の圧力は建築の比ではないのです。

１.5.3.3. 複雑性：量が増大すれば質が複雑化する

建築は規模が大きくなっても部品の種類はさほど増えません。同種の部品を数多く使って規模的に大きくしていくのです。体積が 10 倍になっても、部品の数が増えるのみで部品の種類はほとんど変わりません。
一方、ソフトウェアは似通っている部品があれば共通化して行数を圧縮するので、行数が増えるということは、部品の種類も増えているということです。
つまり建築は体積が増えても質的には大差ないのに対し、ソフトウェアは規模が大きくなれば質的にも幾何級数的に複雑になっていくのです。

> おそらくソフトウェアシステムは、人間の作り出したもののうちでもっとも複雑なものだろう（ブルックス「人月の神話」より）

１.5.3.4. 同調性：環境に依存する

家を建てるときにも、北側斜線とか道路斜線とかその家を建てる環境に基づく様々な制約があります。しかし、ソフトウェア開発において、そのソフトウェアが動く環境から受ける制約は建築の比ではありません。インフラ、OS、ミドルウェア、既存システムなど様々な外部要素に合わせなければならないのです。

本節ではシステム開発を建築と対比しましたが、自動車、家電、服などの他の製造物と比べても、ソフトウェアほど視覚化しにくく、柔軟で、複雑で、環境に依存するもの製造物は他にありません。

1.6 ブルックスの予言は正しかったか？

1.6.1. 銀の弾などない再発射

さて、「今後 10 年間でプログラミング生産性の大規模な改善をもたらすような、ソフトウェアエンジニアリングの発展は一つもないであろう」というブルックスの予言は正しかったのでしょうか？

ブルックスは、1995 年に「人月の神話」20 周年記念増訂版（注）を出版し、その中に「銀の弾などない」も収録しました。この増訂版には、「人月の神話」「銀の弾などない」について 1995 年時点でどのように考えるかを述べた章「銀の弾などない再発射」も追加されており、その中で「銀の弾などない」の予言が基本的に正しかったということが述べられています。

> 1986 年と 1987 年にあれほど声高に推進された秘策というものが、結局、言っていたほどの劇的効果をもっていなかったことに気づかずにはいられない。（「銀の弾などない再発射」より）

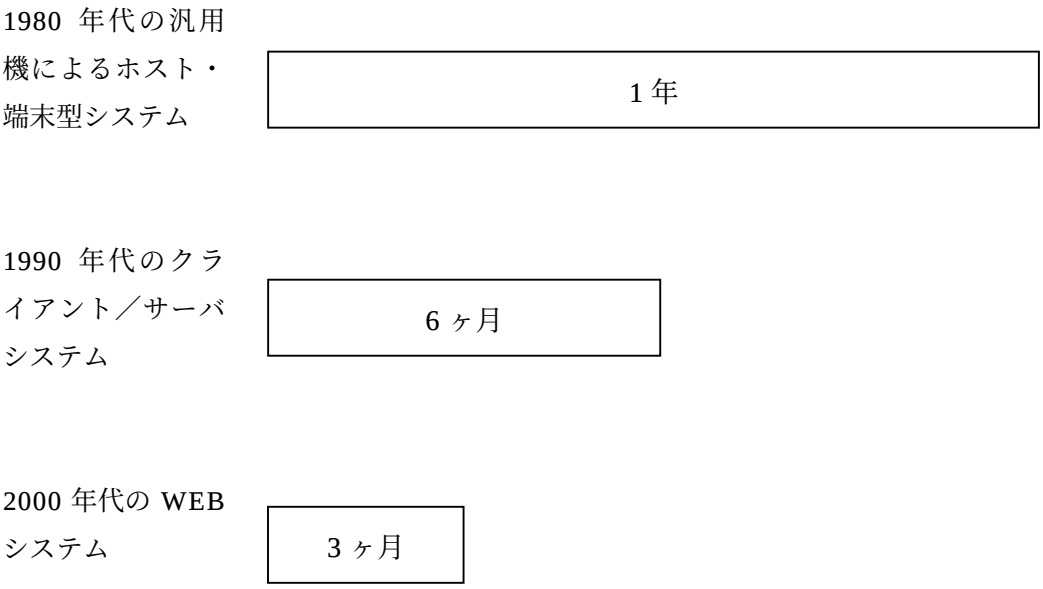
確かに「1986 年と 1987 年にあれほど声高に推進された秘策」が、人工知能、エキスパートシステム、自動プログラミングであるとするなら、それらはほとんど効果がありませんでした。また、ビジュアルプログラミング、オブジェクト指向であるとするなら、その効果は限定的であり劇的ではなかったと言えるでしょう。

> （注）増訂版は日本では 2002 年に「人月の神話[新装版]」としてピアソン・エデュケーション社から発行されました。

1.6.2.　ブルックスが予言できなかったこと

1980 年代までは確かにブルックスの予言どおりに進んでいました。しかし、1990 年代に入り、不思議なことが起きました。

システム形態が、汎用機によるホスト・端末型システム、クライアント／サーバシステム、WEB システムへと移行する過程で、1 プロジェクトの開発期間が下図のように大幅に短縮されたのです。



開発期間のみ見ると急速に開発コストが低下したかのように見えます。これはブルックスの理論では説明できないことです。1990 年代から何かが変わりました。システム開発コストを強引に圧縮する何らかの力が発生したのです。

1986 年に「銀の弾などない」を書いた時にブルックスが予言できなかったことで、極めて重要なことが二つあります。一つはマイクロプロセッサの発明とその驚異的発達、もう一つは経済のグローバル化による標準化・低価格化・短納期化です。この二つについて考察する前に、少し寄り道し、「万人向けでなくなった特効薬」について次章で解説します。