# Engineering and SoftwareEngineering

***Michael Jackson***

【概要】

「ソフトウェア工学」という言葉には、様々な意味がある。

主な意味の1つには、特に重大なアプリケーションのような、信頼性のあるコンピュータベースシステムの信頼できる開発である。

これは解決された問題ではない。

ソフトウェア開発の失敗は、膨大な経済的損失と多くの被害に密接に関わっている。

こうした失敗のいくつかは、プログラミングに起因する可能性や、極限られた感覚(与えられた形式仕様を満たすためのプログラムの欠陥)の違いによるものかもしれない。プログラマの多くは、その他の考え方を持っていると考えられるため、適切な判断である。

それらの根本は、プログラムの正確性の問題にあるのではなく、ソフトウェア工学の問題にある。

有名な 1968 年の協議会は、ソフトウェア工学は「確立した工学分野で古くからの慣習である理論的基礎と実用的な規律の形式」に基づくべきだという、意見によって動機づけられている。

しかし、「ソフトウェア工学」という言葉が流通する 40 年後は、未だにただ漠然と、さらに大部分は実現されない願いにすぎないことを示す。

この期待外れの主な 2 つの原因は、すぐにはっきりする。

1つめは、ソフトウェア開発の広すぎる分野は不十分に特化され、そしてその結果として、信頼性の高い工学の成功の必須である普通の設計のレパートリーが発展しなかったことである。

2 つめは、ソフトウェアに対して構造設計と形式的な分析方法の間の関係が、めったに有益な相乗効果の 1 つとなっていないことである。

あまりに頻繁に彼らの理解できる範囲であるべき両者の強みを奪う、理解不能で相互不信にある競合している独断的な考えとの境界線を明確にしている。

本論文では、これらの原因とその影響について説明する。

ソフトウェア開発の一般的な方法であろうとなかろうと、1968 年の幅広い願望では予測が難しいことも、最終的には満たされるだろう。; 過去の失敗の理解は、確実に将来成功するための必要条件である。

キーワード:

副作用、コンポーネント、コンピュータベースシステム、考案(contrivance)、フィーチャ、形式的分析、標準、操作原理、ラジカル(radical)、特殊化、構造化

**1. Software Engineering Is about Dependability**

**ソフトウェア工学とは、おおよそ信頼性を表す。**

「ソフトウェア工学」の望みは、ソフトウェア開発手法と理論的基礎が確立した工学分野をモデルとすべき、幅広く適用できる信頼を表現することである。

もちろん、それらの分野の実績は、決して完璧ではない。

スペースシャトル「チャレンジャー」の破損、タコマナローズ橋の落橋、そして金属疲労が原因のコメット1墜落事故は、最もよく知られた多くの工学の失敗の一部にすぎない。

しかし、事の是非はさておき、これらの失敗は、常に長期実績上の出来栄えが改善に向かっている局所の欠陥として見られたものである。(改善に向かう途中にある失敗＝改善がされていれば、失敗の出現頻度が格段に抑えられる)

ソフトウェア事業の製品の失敗は、例外というよりも、規則のように見える。

皮肉なことに我々は、Windows を使用する際のフラストレーションを、信頼性の高い現代的な車を使っている時の満足感と比較する。

より深刻的なレベルで、ソフトウェア工学の欠陥は、よくクリティカルシステムの失敗に大きく影響する。

ソフトウェアのアップデートが接続したデータ収集システムをインストールした時、原子力発電所の制御システムは、原子炉を停止する。

1985 年~1987 年に、放射線療法機器セラック 25 のソフトウェアの欠陥は、恐ろしい結果で数人の患者へ放射能の過剰投与を引き起こした。

おおよそ 25 年後、近代的な放射線療法機器は、多くの非常によく似た事件に従事した。

セラックの大惨事に関与するソフトウェアの欠陥は、このような同時代に存在した事件の多くの主要な要因の一つであった。

Risk Forum でまとめられたこれらと、非常に多くの似たような失敗は、ソフトウェア工学製品の中で信頼性を欠いたということの証拠である。

残念ながら、我々は驚かなかった。(ひょっとしたら我々はもはや、これらの失敗によって驚かなかったことにすら)

ソフトウェアエンジニアとして我々は、大志の中で、一番の信頼を置く必要がある。

**2. Software Engineer's Products Is Not the Software**

**ソフトウェアエンジニアの副産物は、ソフトウェアではない。**

共通慣習とは、現実世界と人間世界のことをコンピュータベースシステムの「環境」だと言う。

この慣習は、著しく誤解を招く恐れがある。

「環境」と言う言葉は、観念的に周囲の現実世界と人間世界が、悪意なくソフトウェアが適切に機能するか、全く機能しないかに影響を与えることを意味する。

もし環境が、適切な温度と湿度、さらに少しも地震が起こらないことを条件としていたら、ソフトウェアは、世界からのインタフェースとは無関係に、課題をどんどんこなすことが出来る。

これは、真実からは程遠い。

ダイクストラは、プログラマの真の対象は、コンピュータによって実行され、プログラムによって生じた計算結果である、と意見を述べた。

コンピュータが物理世界と人間世界と接触するコンピュータベースシステムでは、我々はさらに踏み込む必要がある。

そのようなシステムに対し、ソフトウェアエンジニアの真の対象は、コンピュータの外の世界で、ソフトウェアによって生じた振舞いである。

システムの目的は、確固として問題世界に位置付けられている。

つまり、物理世界と人間世界の部分は、直接間接を問わず相互に作用する。

ソフトウェアエンジニアは、ソフトウェアによって制御され、生じた動作が真の研究課題であるので、開発途中のシステムの問題世界と密接に結びついていなければならない。

放射線治療システムの出来不出来は、ソフトウェアを調べるのではなく、コンピュータの外部の影響を観察、評価することで判断できなければならない。(not A, but B)

患者は、正確に所定の位置に向けられ処方された放射線の放射線量を受けるだろうか？

特定の患者が、時折取り違えはあるのだろうか、または戸惑いの原因となるだろうか？

効果的に機器は使われているだろうか？

つまり、放射線治療システムの開発者は、以下についての関心がなければならない。

・詳細な特性と治療機械の装置全ての振る舞い

・治療中の患者の体位と制限

・放射線位の行動、癌専門医の処方箋、患者の振る舞い、必要性と脆弱性

・その他の全てのシステムでおおいに貢献すること

**3. The Software and Its Problem World Area Inseparable**

**ソフトウェアとその問題世界領域は、切り離せない。**

20 年前ダイクストラは、ソフトウェアとその問題世界の間の親密な関係は、それらの間の形式なプログラム仕様を間に入れることによって分離されるべき、と出張した。

「機能仕様書の選択(そして、それらを書きとめるための表記法)は、決して曖昧さがあるかもしれないが、それらの役割は明快である。:

2 つの異なる関心事の間の論理的な「ファイアウォール」として作用することである。

1 つは、「快適性(使いやすさ)の問題」である。すなわち、仕様を満たしているエンジンかどうかの問題は、我々の理想とするエンジンであるかどうかという問題である。;

もう 1 つは、「正確性の問題」である。すなわり、仕様を満たすエンジンをどのように設計するかといった問題である。

2 つの問題は、正確性の問題のための記号処理と、快適性の問題のための実験と藻類学(psychology)によって最も効果的に取り組まれている。」

すぐに興味をそそる議論は、試験(検査)中耐えられない。

エラトステネスのふるい、3 次元の点の集合の凸包、求められるファイアウォールや GCD をそのような問題として考えても、それらは現実よりも明らかである。

ソフトウェア開発者の要求される技術と知識は、プログラミング言語意味論で予測された形式的な記号処理に限らない。

それらは、関係のある数学の問題世界と、十分な多数の定理に熟知していることも含む。

最初の 1000 個の素数を表示するためのプログラムの候補開発者(これから作る人)は、素数の考えと、整数の乗算と除算、そして整数自身のことを知らないことによって、最終的に資格を奪われる。(=前提となる知識がないと駄目)

決してプログラムとその問題世界の間のファイアウォールとして機能しないプログラムの仕様書は、必然的に関係する問題世界の一部と織り交ぜられ、それとなくプログラムの世界でプログラマの中で常識とされている予備知識を当てにしている。

ファイアウォールの考えは、最大の目標が物理世界との相互作用にあるコンピュータベースシステムにおいて、なおさら明らかに機能しなくなる。

それぞれのそのようなシステムに対する問題世界は、非公式の問題領域の特定の異機種環境の組み合わせである。

そのようなシステムのソフトウェアに対するすべての実行可能な設計で、ソフトウェアコンポーネントと問題世界領域の振舞いは、密接に結び付けられる。

システムの実行制御の中心は、それぞれの粒度とレベルで、システム全体の間を行ったり来たりする。

ソフトウェアコンポーネントは、コンピュータの内部で、ソフトウェアのインタフェースにおいて互いに相互に作用する。

しかし、それらは問題世界を通して互いに影響し合う。

ソフトウェアコンポーネントは、プログラムの変数に注意を払うだけでなく、コンピュータの外部の問題領域の状態にも注意を払わなければならない。

それ故に、問題領域は効果的に、ソフトウェアコンポーネント間のさらなる相互作用経路を導入することで共有変数の機能を果たす。

問題領域の振る舞いとソフトウェアの振る舞いを明確に区別するシステムの明瞭な形式仕様は、容易く作れない。

**4. A Computer-Based System Is a Contrivance in the World**

**コンピュータベースシステムは、世界の考案(工夫)である。**

伝統的な工学分野の製品は、物理学者であり哲学者であるマイケル・ポランニーが「考案」と呼ぶものの例である。

自動車や振り子時計、または装置や乗り物は、「特殊な考案」である。すなわち、世界で制限されたコンテキストで特定の人間の目的を達成するために設計された物理的な人工物の組み合わせである。

コンピュータベースシステムは、まさに同じ意味の考案である。

考案は、それぞれに特有の特性と振る舞い、特有のコンポーネントの構造を持っている。

コンピュータベースシステムにおいて、それらの特有のコンポーネントは、ソフトウェアの制御下で互いに影響し合っている問題世界領域である。

コンポーネントは、相互作用が考案の目的を実行するため、作られる。

コンポーネントによって実行される目的は、考案の「操作原理」である。: 言い換えれば、それがどのように働くか(=操作原理)という説明である。

例えば振り子時計において、重さによって作用する重力は、バレルに回転力を発生する。;

手で歯車列を回すことで発生するこの力(回転力)は、歯車列によって伝播する。;

振り子のそれぞれの振れに対し、1 つの歯によって回転する脱進機の回転も伝播する。;

それ故に、手の回転は、振り子の揺れの数に比例する。;

大体一定の割合で振り子は揺れるので、手の角度位置を変えることは、振り子は選ばれた基準点に定められたため、どのくらい時間が経過したのかを意味する。

そのような機械の操作原理と設計は、ポランニーが「考案の論理」と呼ぶものを示す。

これは、物理学または数学を単純化できない、そしてそれらとは異なる何かである。:

それは、人間の目的と設計された考案によって達成された目的がどの程度の直感的な理解を伴うことかの理解である。

何はさておき、それは、具現化された創造と視覚化の人間の能力によって支えられた、考案の練習である。

それ自体は正式ではない。しかし、それは、正式な対象に(すなわち、例えば、自然現象の理解または数学の法則の概念で)適用することが出来る。

ナターシャ・マイヤーは、生徒にタンパク質の折り畳みを、腕と手で折り畳み過程を身体的に表現することによって説明し、解説した素晴らしい科学の教師について説明する。

「この過程では、科学者の体は、他の人に知識を伝え、他の人が学ぶための道具となる」と彼女は書いた。

ここでいう知識とは、本来、タンパク質の折り畳みの作用の仕組みに対する「感覚」である。

## 5. General Laws and Specific Contrivances

### 一般法則と特有の考案

自然法則と工学の間の重要な差異は、自然界の法則と考案の論理の間の差異に表される。

科学者の熱意は、普遍的な、または少なくともとても一般的な性質の法則を発見することである。

実験は、考案された法則に関係のないすべての複雑さを除くことによって、出来るだけ厳密に推定される法則を検査するように仕組まれる。

従って、実験結果の値の普遍性は、重要でないと見なされたあらゆるものからの分離によって決まる。

化学実験では、化学実験装置は完璧に他の物質が入っていない状態でなければならない。また、使用済みの化学薬品は、出来るだけ他のものを混ぜ合わせていない状態でなければならない。

電気的実験では、浮遊容量またはインダクタンスを考慮しないと言った非常手段がとられるかもしれない。

理想的に、実験の対象である推定される法則の影響だけが観測されるべきだ。:

全ての他の法則の影響は、ある意味で、一定に保証される、または相殺されていなければならない。

契約によって設計された考案は、技術者の指令書によって暗示された限定されたコンテキストでしか使えないように設計されている。

振り子時計は、適切な強さの重力場でなければ正常に動作することが出来ない。

重力に対して直立位置を維持しなければならず、また全体として加速度の影響下にあってはならないため、振り子時計は、海で船の上で正常に動作することが出来ない。

振り子時計は水中や油中に沈められると、振り子運動に対する抵抗がとても大きく、また重みの影響が、液体の上へ向かう推進力によって極端に減少してしまうため、動作することが出来ない。

たとえ選択したコンテキストが制約のあるものでも、その時は、ありのままを受け止めなければならない。

考案は、実用的に使用することを目的としており、選択したコンテキストで実際にうまく機能しなければならない。

設計は、前もって影響が無視または過小評価されていたいくつかの未知の自然法則、またはいくつかの現象に悩まされるかもしれない。

これらの不都合は、見当違いの判断をしたり、ないことを望まれたりすることはない。

失敗や欠陥が明らかにされた時、エンジニアは、設計を改善することによって、問題を解決するための方法を見つけなければならない。

コンテキストでさらなる制約を定めることは、まれにしか実行しない、そして決して望まれない。

例えば、早期に振り子時計が冬に早く進み、夏に遅く進むことが明らかになった時、確実に、1年のうちの 1 シーズンの仕様に制限するための可能な措置をとることはない。

工学の解は、欠くことのできないものであり、振り子の揺れの軸と振り子の重心の間の一定の距離を維持するような温度補償の振り子から入手することが出来た。

もちろん、考案とその構成部品は、自然の法則に逆らうことが出来ない。

しかし、自然の法則は、考案の振る舞いに対する制約の最外層だけを提供する。

この最外層は、制約のあるコンテキストの利用法に起因する制約の内装である。

自然の法則は、ひょっとしたら予測によってエンジニアに、どのように月で時計が振る舞うかということを予測することを許す。しかし、その予測は、時計が月で動作することを対象としていないため、無意味である。

制約のあるコンテキストの制約の中には、考案のエンジニアの設計に起因する制約のさらなる層がある。

要素と相互関係は特に、考案の操作原理を具体化するために、制約のあるコンテキストの中で適用するので、自然法則によって不確定なものを残された環境の中で、エンジニアによって形作られる。

それは、明確な人間の目的であるコンテキストの制約と、エンジニアのコンテキストをいいことに、科学だけでなく工学を作ると言う目的を果たすための考案の形式である。

工学を自然科学の応用と見なすことは、誤解である。

もしそうだったとしたら、我々は、橋とトンネルを設計するための優れた物理学者を目指すだろう。

工学は、物理学に還元できないため、現実には我々は違う。

ポランニーの言葉の中にこんなものがある。:

「工学と物理学は、2 つの異なる科学である。

工学は、機会の操作原理と、それらの原理に耐えるいくつかの物理特性の知識を含んでいる。

他方では、物理学と化学は、機会の操作原理の知識を含む。

従って、完全な物体の物理・化学トポロジーは、それがマシンであるかどうか、もしマシンであるとしたら、どんな目的で、どのように動くのかを我々に教えてはくれない。

マシンの科学的調査と物理調査は、マシンの既に確立された操作原理に関連している下に行われるものでない限り、無意味である。(=原理に根差していなければ意味がないということ)」

**6. The Lesson of the Established Branches**

**確立された分野の教訓**

40 年以上前、ソフトウェア開発の発展と成果に対するとてもたくさんの不満と、「ソフトウェア危機」の話がたくさん存在した。

その当時は、コンピュータベースシステムの分野で、物理問題世界の中心的役割と主な役割は、率直な興味の焦点ではまだなかったけれども、それにもかかわらず、多くの人々はエミュレートするためのモデルとして、確立された工学分野に目を向けた。

1968 年と 1969 年の有名な NATO のソフトウェア工学会議の明確な動機はこうだった。:

「1967 年の後半に、研究グループは、ソフトウェア工学についての実用的な会議の実施を推奨した。

「ソフトウェア工学」と言う言葉は、工学の確立された分野の伝統的な実用的な規律と理論的基礎の種類に基づくために、ソフトウェア製品の必要性の意味で、刺激的であるとして意図的に選ばれた。」

確実に、参加者は無関心ではない。

1968 年の集合で、ダイクストラは、議論で次のように言った。:

「信頼できる人々のこのグループでの、ソフトウェア障害の存在の全員の告白は、数年間で私が思ってきた、とても刺激的な経験である。なぜならば、欠陥の告白は、改善のための第一条件であるからだ。」

けれども、ともかくも、研究グループの明確な動機が、発表や議論でちょっとした役割を果たした。

ＭＤ マキロイによる招待公演は、もしかすると唯一の例外である。

マキロイは、入出力、三角関数またはコンパイラで使うための記号表のためのソフトウェアコンポーネントのカタログを提供するだろう、部品工業の必要性を主張した。

しかし、この提案は、それが行われた単一のセッションを越えて決して広範でない議論を、刺激した。

会議の参加者は、実行(エミュレート)するために勧められた理論と実践を調査しなかった。

もし、このやり方で控えめな始まりをしていたら、彼らは確実に確立された分野から学ぶべき最も広く、最も簡単で最も明白な教訓を認めるだろう。

「確立された分野」ではなく、単なる複数から成る「確立された分野」は、ある分野に特化したものを一度に見られることが出来る。

土木技師は自動車を設計しないし、電力技師は橋を設計しない、そして航空技師は化学プラントを設計しない。

完全にこの教訓を学び、生かすための我々の歴史的失敗は、「ソフトウェア工学」と言う言葉の永続性によって良い例となった。

ソフトウェア工学は、確立された分野が「具体的な工学」の単一の領域を構成するという仮定の下でのみ、１つの学問分野としてなり得る。

それどころか次のようなことがいえる。:

彼らは違った。

彼らの特殊化にとりわけ依存する議論の余地のない分野で、我々がエミュレートしたいと望んだことは、とても成功した。

**7. Specialisation Has Many Dimensions**

**特殊化は、様々な側面を持っている。**

ここで重要な意味で、専門分野は、個人目標における集中した個々の１つの焦点ではない。

それは、長期間を通して永続的な類似性、科学的試みまたは技術的試みの特定のフィールドの共通の知識を拡張し、有効に使い、維持し、開発するということの焦点である。

特殊化は、変化する要求や機械への対応において生じ、発展する。さらに、特殊化は、多くの異なる連動する側面と、フィールドの側面と横断的側面に焦点を合わせる。

確立された工学分野は、とても高度なこのプロ節を説明する。

工学成果物による特殊化が存在する;

すなわち、自動車、航空、海軍、化学工学の工学成果物による特殊化である。

問題世界による特殊化が存在する;

すなわち、土木工学と鉱山工学の問題世界による特殊化である。

さらに、要求による特殊化が存在する;

すなわち、生産技術、生産工学や輸送工学の要求による特殊化である。
理論的基礎の中で特殊化が存在する;
すなわち、制御工学や構造工学の理論的基礎の中で特殊化が存在する。
工学製品の分析で生じる数学の問題を解決するための手法の中でも特殊化が存在する;
すなわち、有限要素分析やコントロールボリューム分析といった分析手法
大きなシステムで利用するための設計されたコンポーネントで特殊化が存在する;
すなわち、電気モーターや内燃機関や TFT 液晶画面といった大きなシステム
工学技術と原料で特殊化が存在する;
すなわち、溶接といった技術と、鉄筋コンクリートや電導性プラスチックといった原料で
そして、その他の側面でも、特殊化が存在する。
それらの特殊化は、任意の単純な階層構造に分類されない。
多くの側面で、彼らは、全ての粒度で領域を重ね合わせること、そして、最も実用的なことから最も理論的なことまでの全ての関心事や、19 世紀前半の高圧蒸気エンジンでボイラー爆発をなくすための科学的な根拠として定めている熱力学の発展のような、理論的な話題への一種の伝統工芸品である工学の実践のようなたぐいのものからのすべての関心事に焦点を合わせる。

**8. The Key to Dependability Is Artifact Specialisation**

**信頼性の鍵は、成果物の特殊化である。**

提示する特殊化の多くの側面のいずれかを除外する理由はないため、世間に認められたエンジニアの間で誰かを選ぶ際には、我々はエミュレートする義務がある。;
しかし、基本的であり、不可欠であるとして、工学の特殊化の 1 つの側面を考慮するための、非常に実用的な理由がある。
我々が「成果物の特殊化」と呼ぶものである。(↑の理由と言うのは)
つまり、特定のクラスの成果物の構造と設計での特殊化である。
1 人のエンジニアの設計のコンポーネントは、他のエンジニアが専門とする成果物であるかもしれないことと、同じ成果物は、別の成果物クラスを含むコンポーネントとして思われるかもしれないので、成果物の特殊化は、潜在的に複雑な構造をもつ。
この構造の中で、成果物の特殊化は、十分に専門的な工学グループが、どこにグループの外部の顧客に配信するために設計された彼らのグループの成果物に対して存在していようとも、特定され得る。
専門家のエンジニアの成果物は、完成した成果物であり、また、値の合計や経験、アフォーダンスによって影響を受けるあらゆる人とユーザを提供するので信頼できる。
抜け道も免責事項もない。
これが、成果物の特殊化の要因と基準のいずれにもおける、工学の責任の本質である。
効果的な成果物の特殊化は、対処し易い選択肢ではない。
最も成功したひな形では、それは専門家の団体の中で、個人とグループによって集中的、かつ

持続的な研究への取り組みを要求する。(=成功するためには、熱心な研究が必要)

ウォルター・ヴィンセンチは、「エンジニアは何を知り、それをどのように知るのか」という彼の本で、1915 年から 1945 年までの航空工学における 3 つの顕著な例を説明する。

1 つ目の例は、プロペラ設計の問題への取り組みであった。

最も初期の航空機は、2 枚の翼のプロペラで駆動していた。

そのようなプロペラの設計は、プロペラの翼は航空機が飛んでいる方向に動作しているだけでなく、プロペラの回転軸の周りをまわっているため、空気力学的に、航空機の翼の設計よりも複雑である。

この問題に対処するために、ＷＦ デュランはとＥＰ レスリィは、1915 年から 1926 年の長年にわたり共同作業をし、150 のプロペラの設計に関する風洞実験と詳細な分析的研究を実行した。

2 つめの例は、安定板または水平尾翼または期待のフレームに対して、金属膜をパネル貫通するリベットの問題と関係があった。

第一の金属膜が航空機に対して実用的になった時、それはドーム型のヘッドを持つリベットによってフレームを固定された。

(研究によって)すぐにドーム型のヘッドが著しい空気抵抗を引き起こすことによって、性能を低下させるということが、明らかになった。

それゆえに(空気抵抗を抑えるために)、パネル貫通するリベットは、リベットのヘッドがさら頭の外形を持ち、期待の表面上に突出していないことが望ましい。

外板は、一般的に厚さが 1mm に達しないほどに薄かった。;

なくてはならない皿穴は、外板を弱くしたり、または操作のストレス下で緩みを引き起こしたりしないことを実現されなければならなかった。

この問題に対する満足のいく解決策は、1930 年から 1950 年までにいくつかの航空機製造会社の中で 20 年間の提携をとった。

3 つ目の例は、ソフトウェア工学で疑いの余地のない直接的に同等のものかもしれない。

1918 年の航空に関するパイロット達は、航空におけるある非機能要求に対する強い願望をはっきり述べるために始めていた。:

彼らは、飛ぶために安定しており、信頼でき、動作が予想でき、要求を満たしているような優れた飛行性を持つ航空機を望んでいた。

その疑問は、生まれた:

・パイロット達は、本当に何を 3 つの品質品質要求としているのか？

・どのような設計された航空機の行動性が、これらの品質を保証するだろうか？

・どのように設計者は、それらの行動性を達成するのか？

それらの疑問における専門家の研究は、20 年以上後に実施された。

1940 年に関する答えの主な要素は、当時の標準の設計の航空機のために確立した。すなわち、言い換えれば、横方向に対称性を持つ航空機、真っ直ぐな前縁、そして後方に垂直安定版と水平尾翼を持つ単一の翼の設計のために確立した。

答えは、必然的にとても技術的であった。
彼らは、確立された標準的な設計を適用した。;
彼らの設計の観点から表現された。;
そして、彼らを探すことは、四半世紀を越えて、計装技術者とパイロットとデザイナの専門のある分野に特化した努力を要した。

**9. Normal and Radical Design**

**一般的な設計と抜本的な設計**

そのような航空機の特殊化の成果の表現と、直接表現は、関連付けられた「一般的な設計」である。:
すなわち、言い換えれば、特殊化の基準となる工学デザインの実践と開発する標準的な設計の製品が、設計における表現である。
ヴィンセンチは、このように一般的な設計の特性を示した。:
「エンジニアは、最初に、どのように問題となっている装置が動くのか、何が慣習のフィーチャであるか、その上もし、そのような流れに沿って適切に設計されていたとしたら、目的のタスクを果たすために十分あり得そうなことが分かる、ということを知る。」
彼は、1945 年以前の時期に、航空エンジンの設計の実践によって、一般的な設計を開設する。:
「例えば、ターボジェットに先立ち、通常の航空エンジンのデザイナはそれ(解説)を、仮にエンジンが、ガソリンを燃料とし、4 サイクルの内縁の周期によるピストン駆動であるべきことを認めることである、と思った。」
高出力エンジンのためのシリンダの配置は、液冷式の場合、線形のバンク角の時にラジカルを、そして空冷式の場合、ラジカルを与えられたとしてみなされるであろう。
それでまた、余り明らかでない他のフィーチャだろう。(例えば、発言力とスリーブ弁に対する他ペット) So also would other, less obvious, features (e.g., tappet as against, say, sleeve valves.)
デザイナは、この性質のエンジンを良く知っていた。そして、成功の長い伝統を経験として持っていることを確信した。
大抵、制限内で非常に要求の厳しい設計の問題は、減少した重量の方向と、燃費の向上または出力の増加またはその両方の改善の 1 つであった。
契約によって、抜本的な設計においては次のようである。
「装置がどのように配置されるべきか、または装置がどのように動作するのかは、ほとんど知られていない。
デザイナは、以前にそのような装置を見たことがなかったので、成功の推測が出来ない。
問題は、さらなる進展を保証するために、十分良く機能するだろう何かを設計すべきことである。」
設計が、これまであらゆる点で、完全にそして徹底的に抜本的であったことはない。なぜならば、

デザイナの予備知識と以前の経験は、必然的に、潜在的にタスクに関係があり、有益な何かをもたらす。

例えば、カール・ベンツは、初めて完全な自動車の設計に対する特許を求めた前に、ガソリンを燃料とするエンジンの設計で、成功裏に取り組んだ。

ベンツの設計の詳細と、正確なレプリカの写真(2 つのレプリカのシリーズは、20 世紀後半に製造された)は、Web 上で簡単に入手できる。

車は、3 つの車輪と、放射状の金属スポークと、硬いゴム製タイヤを持つ。

ドライバーは、ラックアンドオピニオン式の装置を通して、逆向きの鉛直方向に回転させられた前輪を、クラックハンドルによって操縦する。

車は、鉛直方向のクランクシャフトに設置されており、後部座席に後輪の間の車の座席を取り付けられた広々としたクランク室で単気筒 4 サイクルエンジンによって動力を供給される。

広く開いたフライホールは、クランクシャフトに取り付けられたエンジンのフライホイールを起動するためには、手によって回転させる。

入力電圧は、かさ歯車、クラッチ・作動装置としても動作するベルト駆動、そして最後にはそれぞれの後輪へのチェーン駆動を通して、車輪に伝播される。

制動は、中間軸に作用するハンドブレーキによってもたらされる。

この設計は、多くの新しいアイデアを具体化した。そして、疑いようもない天才的作品であった。

実用的な工学成果物として、ヴィンセンチの基準を満たした。(さらなる発展を保証するのに十分な働きだった。) しかし、それは、多くの問題を持っていた。

多くの顕著な失敗は、ステアリングの設計にあった。

単一の前輪は、滑らかな表面以外で乗り心地が不安定になった。そして、車両別にクランクのとても小さい半径は、ドライバーにほんの少しの操作しか与えなかった。

制御は、長いクランクの処理によって改善されることが出来た。しかし、単一の前輪は、深刻な設計の問題であった。

問題は、ベンツが採用した単一の前輪駆動と考えられている。なぜならば、彼は、2 つの前輪のステアリングの問題を解決することが出来なかったからである。

それぞれの前輪の軌道を確保することは、進行路と接線方向である。内車輪の回転角は、外車輪の回転角よりも大きくなければならない。

これを達成するために必要とされる構造は(すなわち、現在では一般に「アッカーマンのステアリングの形状」として知られている)、馬車の作り手によって 70 数年前に特許権を取得された。そしてまた、フランスで設計された上記の乗り物の中で、10 年前に使われていた。しかし、ベンツと彼の同年輩であるゴッドリーブ・ダイムラーのどちらもそれ(必要な構造)を知っていた。

5 年後、ベンツは、原理を再発見した。さらに、1891 年に、特許を取ることに成功した。

彼の 1893 年の車であるヴィクトリアは、4 つの車輪を持ち、アッカーマンのステアリングを組み込んでいた。

**10. Artifact Specialisations in Software Engineering**

**ソフトウェア工学での成果物の特殊化**

ソフトウェア工学は、確実に多くの目に見える特殊化がある。

コンピュータサイエンスに属すると見なされることが出来る理論的な領域で、多くの機能している特殊化が存在する。 しかし、ソフトウェアベースのシステムの工学のためのアプリケーションを直接利用する。:

例えば、同時並行性と複雑さの理論。

関連性のある手段と科学技術の中で、多数の特殊化がある。:

例えば、ソフトウェア検証、モデル検査、形式仕様言語、そして Web アプリケーションを開発するためのプログラミング言語。

成果物の仕様書も存在する。しかし、とりわけ問題世界の成果物に対してであり、必ずしも単なる概要ではないが、依然として、なだめるように人間世界と物理世界の不安定さと複雑ささせる要因に結び付かないままである。

フレデリック・ブルックスは、オープンソースムーブメントで、コメントの中できちんと成果物のこのクラスを特徴付けた。:

「Linux コミュニティで、市場の顕著な成功は、作り手もユーザであるという事実からすぐに推進するように思える。

それらの要求は、それ自身とそれらの作業から生じる。

それらの願望する基準と経験は、自身の体験から不意に来るものだ。

全ての要件決定は、潜在する。それ故に、巧妙に行われる。

私は、作り手が自身の作った物のユーザでない時、そしてユーザのニーズの受け売りの知識しかない時、オープンソースは同じく有効であるかどうかは、甚だ疑問に思う。」

「あなた自身のドッグフードを食べる」とう見事な主義は、開発者が顧客でもある、または簡単に完全な信念を持って顧客の役割を果たすことが出来る時、ぴたりと当てはまる。

だから、コンパイラやファイルシステム、関係データベースシステムマネジメントシステム、SAT ソルバ、そしてスペルチェッカーでさえに効果を発揮する効果的な特殊化が存在する。

それらの成果物の特殊化は、開発者の創造力豊かな視野の中でよく破綻する。そして、物理世界の賑やかで途方もない無秩序は、無事に見えないところにある。

同じことは、放射線治療または自動車システム、または原子力発電所または電力供給網を制御するためのシステムに言うことが出来ない。

それらのシステムと、それほど重要でないが、それでも重要な多くの他のコンピュータベースシステムがどの程度であるかはいまだに明らかでないことは、システムの長い歴史を越えて確立した工学分野を特徴付けられた、徹底的な特殊化から恩恵を得始める。

特殊化自体が確立することでさえ、十分な一般的な設計の進化は、何年も時間がかかりそうである。

カール・ベンツの発明は、1905 年までに、国際的な産業を引き起こした。

しかし、1920 年代まで存在しなかったベンツの自動車は、一般的な設計の対象になっているということが出来る。

**11. Artifact Specialisation Needs Visible Exemplars**

**成果物の特殊化は、普及している原形が必要である。**

多くの文化的で社会的で組織的な、ソフトウェア工学にとって成果物の特殊化の発展と出現による障害が存在する。

ある障害は、一般的な性能と呼ばれる。また、特定の類似の切望と呼ばれるかもしれないものである。

雑誌や会議で具体的な実システムの説明は、めったに宣言されない。

せいぜい、用意された事例研究が、さまざまな論文を支持するために、概要を説明された実例として提示されるかもしれない。

我々は、あるシステムの詳しい研究から学ぶことが出来るほとんど学ぶ価値がないことを仮定しているようだ。

仕様書の警視は、効果的な成果物の特殊化の進展に対して強く影響を及ぼす。

確立した工学分野の中で、エンジニアの実践は、彼の失敗や特性、確立された一般的な設計についての特定の実際の成果物から学ぶ。

例えば、イギリスの地震工学研究センターは、多くの種類の大きな建物や、橋のような実際の土木建築物の見本を表示しているスライドの大規模な図書館を保持する。

その所蔵品は、主に成果物のクラスに整理される。そして、大学の学部課程と大学院課程のための教育資源として役に立つ。

学生は、理論の知識を身につけることだけでなく、具体的な実際の工学の例の情報により提供された検査をすることでも学ぶ。

それぞれの例は、教えるための特定の教訓と、豊富な分類に存在する。

失敗は、成功よりも重要であるということが、少なくない。(すなわち、もしかしたら、ましてやなおさら)

あらゆる工学部学生は、1940 年のタコマナローズ橋の落橋を保存した有名なアマチュア動画を見せられる。

デザイナであるレオン・モイセイフは、特に垂直変更の方向に向かって、橋の幅員が狭く、浅い道路で風の空気力学的効果を少しも考慮していなかった。

ちょうど毎時 40 マイルの風の中では、道路の鉛直方向の振動は、完全に端を破壊させる大きさに高まった。

つり橋のデザイナは、教訓を学んだ。そして、教訓を守るために責任を認め、そして教訓を後任者へ伝えた。

それはとても実用的な教訓である。

モイセイフの過ちは、繰り返されないだろう。

ソフトウェア工学において、我々は、我々の設計で見つける過ちを繰り返さないと少し確信できる。

成果物の特殊化なくして、十分に実用的な教訓を学ぶことが出来る内部で、構造は全くないかもしれない。

数年後、優れた調査研究であるセラック 25 の経験が、IEEE コンピュータで出版され、そして、[14]の付属文書として、改良された時から掲載された。

調査員は、すべての入手可能な証拠を調査し、彼らの論文は、信頼のできる実用的なソフトウェアとシステムエラーを特定した。

しかし、放射線治療機械のソフトウェアに対する一般的な設計がないことは、ソフトウェアコンポーネントに対する用語体系の欠如により、はっきりと証明された。

電気機械学の機器の一部は、論文に掲載された図表で、規格名を与えられた。:

回転板、分割統治、反射鏡、電子スキャン磁石など。

しかし、ソフトウェアの部品は、規格名がなかった。そして、例えば、「タイラーの偶発事件に対する責任を負わされたコードの中のタスクとサブルーチン」としてプログラムのコードから受け継いだ与えられた名前である個々のサブルーチンに言及する。

必然的に、論文の最後に提示された忠告は、主として利用されていた開発プロセスについて、一般的なポイントに集中した。

忠告は、文書、ソフトウェア仕様書、危険なプログラミング実行の回避、オーディット・トレール、検証、形式的分析、そして、同様な懸念に焦点を当てた。

実用的な忠告は、少しもソフトウェアの成果物自身について構成されることが出来ない。なぜならば、そのような忠告が言及できるための標準的で一般的な設計ではないからだ。

セラック 25 のソフトウェアエンジニアのエラーは、残念ながら、25 年後に繰り返された。

## 12. The Challenge of Software System Structures

### ソフトウェアシステム構造の課題

ほとんどすべての実際のコンピュータベースシステムは、多くの場所で複雑である。

この複雑さは、システムの設計と、やりとりする構造と、説明する構造の主な障害である。

それ故に、一般的な設計の進化の主な障害である。

より正確には、我々は、1 つの構造だけでなく、多くの構造のことを話すべきである。

システムは、多くのフィーチャを提供する。各自識別できる機能の単位は、ユーザの要求と周囲の事情に従って利用され得る。

それぞれのフィーチャは、関連付けられた考案によって自覚されたと見なされることが出来る。

考案のコンポーネントは、問題領域と、システムのソフトウェアの一部、または突出した部分にある。適切に設計された、フィーチャの機能を実行するために互いに作用する構造に整えられる。

フィーチャは、必然的に、互いに影響する。:

すなわち、それらは、問題領域とソフトウェアリソースを共有することが出来る。;

それぞれは、どんなときにも、連携に積極的である。;

そして、それらの要求は、矛盾するかもしれない。

相互作用と連携は、それ自身独特な構造と、徹底した調査と分析を要求すること、そして両方の要求を満たすことを期待される要求に関して、そして実装の単位において、誘発するだろう。

このフィーチャの相互作用問題は、20 世紀後半の電話システムで目立つようになった。自動手脳や発着信制御のような呼処理のフィーチャの相互作用は、驚くべき結果と、ときどき潜在的に弊害を引き起こすと見なされた。

これらの相互作用の構造は、例えば、分割されたフィーチャの構造で使用する種類のクラスで、取り込まれるかもしれない。

フィーチャの相互作用問題は、極めて電話システムのソフトウェアを作ることのコストと難しさを増加させた。

すぐに同じ問題は、非常に種類の複雑なシステムで本来備わっていたことが明らかになった。

全体として、単一の考案と見なされていたシステムは、操作のいくつかのコンテキストを持つだろう。

例えば、事務所を含む建物、店舗やアパートにおけるエレベータを制御するシステムは、用途によって毎時で、そしてその日その日で、異なる需要パターンを考慮しなければならない。

消防隊の支配下での非常時運転、ライセンス機関の代表者のテスト制御下における運用、そして装置の定期メンテナンスをしている間の運用のように、特別な運用のコンテキストも存在する。

もし、装置の故障が見つかったとしたら、システムは、可能であれば、障害を和らげるための処置を取らなければならない。すなわち、例えば、装置の不完全なサブセットを取り出すことによって、またはエレベータのカゴでこれまで移動していた多くのユーザの安全を確保するために応急処置を実行することによって。

それらのコンテキストは、バラバラである必要はありません。:

例えば、障害は、検査の間に見つけられるかもしれない。そして、試験技術者の安全は、確保されなければならない。;

ドメインとなる個々の問題のプロパティ(すなわち、エレベータシャフト、床、扉、エレベータのカゴ、利用者、巻き上げモーター)とふるまいは、開発プロセスで分析されなければならない。

それぞれのフィーチャを実現する考案の設計は、関係のあるプロパティと、設計の中でコンポーネントとして働く問題ドメインの振る舞いを考慮しなければならない。

問題ドメインを具体化するソフトウェアオブジェクトを具体化することは、実行済みのソフトウェアにとって、必須である。

ドメインは、プロパティにおいて、それぞれのフィーチャに対するモデルを持ち、そして、それらのモデルは、組み合わされたいくつかの方法で調整されなければならない。

実装されたソフトウェア自身は、モジュール構造の(実行基盤と仕様するプログラミング言語のフィーチャによって規定された構造により、相互作用するモジュールのパーツ)いくつかの種類を持つ。

物理的な人工物によって共有されないソフトウェアの極端な可塑性は、システムの他の構造とは根本的に異なるための実施体制をゆるす。

それらの構造の間の関係は、そのとき、変換によってとりなされるかもしれない。

これらの構造のすべては、なんらかの形で、保存され、設計され、分析され、調整され、組み合わされる。

タスクは、確実にトップダウン分割または改良のような、単純化した一般化手法によってうまく実行されることが出来ない。

そしてまた、要素の大規模なコレクションにシステムを断片化する還元的アプローチの種類によって、回避されることが出来るわけでもない。

一般に、これは未解決の問題である。

一般的な設計のタスクにおいて、十分な解決策は、長期間にわたり考案されてきた。そして、専門家の実行者によって導入された。

そこには、一般的な設計は適用しない。そして、設計作業の側面と主な部分は、不可避的にラディカルであり、設計者に重荷を感じさせる問題の重点である。

この問題は、多くのコンピュータベースシステムの設計の顕著なフィーチャである。

**13. Forgetting the Importance of Structure**

**構造の大切さを忘れないこと**

コンピュータの動作が遅かった時よりも以前に、プログラムはより小さく、コンピュータベースシステムは、またソフトウェアの世界で注目の的ではなかった。プログラムの構造は、研究の中心であり、重要なトピックとして評価された。

有名な(学術誌の)レター(=Abstract)の中でダイクストラによって提唱されたように、構造化されたプログラミングの規律は、明確に、ソフトウェア開発者は作っていたプログラムを熟知できるべきという信頼によって動機づけされた。

構造の主な利点は、静的なプログラムのテキストと動的計算の間に、非常に明確な関係にあった。

構造化されたプログラムは、計算の進展を理解するための便利な座標系を提供する。

テキストの中のそれぞれのポインタがネストされている中で、座標は、テキストポインタであり、そして任意のループに対する実行カウンタである。

ダイクストラは、次のように書いた。:

「我々は、なぜそのような独立座標が必要であるのか？

理由は、(すなわち、逐次プロセスに内在すると思われるものは)我々がプロセスの進展への関心のみで変数の値を解釈できることである。」

構造化されたプログラムは、はっきりと定義された進化する状態全体の計算をするための簡単な方法にマッピングするコンテキストで、それぞれの状態のコンポーネントの連続した値をセットす

る。

プログラムの構造木が、それぞれのレベルでどのようにそれぞれの字句コンポーネントを表示するかは、すなわち、単純なステートメント、ループ、if-else、または連結といったテキストの中に置かれている。

もし、字句コンポーネントが、何度も実行され得るならば、それぞれ含んでいるループのための実行カウンタがどのように実行するのかは、全体の計算の中に置かれている。

プログラムのそれぞれの地点のプログラマの理解は、構造木の根までずっと至るネスト化されたコンテキストの構造に置かれる。

構造化されたプログラミングを推奨した制御構造の規律を越えて、他の構造の概念は、(すなわち、例えば、クラスタの働きによるプログラミングのナウルの規律や、構造化された抽象概念)人間の理解を支持することによって、プログラムの改善や単純化の目的で、常に研究される。

第一に必要なものは、明瞭な構造に対するものです。

正しさの形式的証明、または厳密な証明は、その後、設計されたプログラムの構造によって導かれた証明の構造のふさわしい副産物になるだろう。

この方針の形式的側面を検討することは、翌年中に鮮やかに達成できたことを示した。

しかしながら、残念なことに、それは徐々に色あせた人間の理解と、単純に人間の解釈に対する興味を失った多くの最も創造的な研究とを重要視することで、とても成功した。

それは、プログラムの構造化を促進するための形式的証明の技術を可能にするために、より魅力的に見えるようになった。

1982 年に関して、ナウルは不満を言っていた。:

「形式化に重点を置くことは、プログラム開発に悪影響を与えることが知られている。例えば、単純な形式化と非形式的な制度の無視などである。

直感的な分かりやすさを強化するためだけに形式的で使われている仕様書は、推奨された。」

プログラムの構造を決定するための小さく、単純なプログラムに対し、段階的な詳細の進行を許可することは、実行可能である。

典型的に、プログラムの目的は、容易に簡潔に、形式的に明記できるような結果を確立することである。

証明は、与えられた事前条件から、この結果を確立するために要求される。

しかし、ナウルの訴えは、正しかった。

確実に、ソフトウェア工学の基礎であるべき人間の理解と直感の印象の原理であり、放棄された一度にいくつかの形式的装置が導入されていることが出来得る粗野な足場ではない。

形式的主義者の活動の成功は、人間の理解の役割の中で、まさに彼らが最も地位が高くなった時、厳密に減少した構造の問題や関心事への関心のきっかけとなる。

絶え間ないコンピュータベースシステムの複雑さの増大は、純粋に形式的な手段で厳密に対処されることが出来ない。

45 年前、見たところプログラムのフローチャートを設計することを理解することにおいて、扱いに

くいような問題は、人間の理解に仕えた構造の導入に従った。(←人間の分かり易いものを導入した)

今そして将来において、構築されているシステムの複雑さを管理し、使いこなすためには、人間の理解を必要とする。

形式的技術は、肝心である。しかし、その技術は、人間的に分かりやすい構造の中で展開されるべきである。(=技術は、人間が理解して初めて意味がある。=人間が理解できるものであるべき。)