2020/5/29

ゼミナールⅠ　レポート

[redacted]

この章では、大きく複雑なソフトウェアシステムの開発の最中に
生じる問題について探求していく。

ソフトウェアエンジニアリングの研究者の目的

ソフトウェア開発を導出し、
効率的で信頼性のあるソフトウェア製品を導く原理を見つけること。

ソフトウェアエンジニアリングとは？

・ソフトウェア開発は工学プロセスなので、ソフトウェアエンジニアリングと呼ばれる。
・大きく複雑なソフトウェアシステムの開発を模索するコンピューターサイエンスの一つ。
・その複雑さは単純比例ではない。システムの大きさにより、指数関数的にエラーが増大することもある
・多くの人の労力が必要とされ、関わる人間が変化することもあるため、コンピューターサイエンスというよりもビジネスマネジメントに大きく関与している

ソフトウェアエンジニアリングにおける問題を理解する。

まずは、大きく複雑なもの（ソフトウェアでない）を
設計、構成する際に生じる問題について考える

ダクトはどんなデザイン？
といった細部に至るまで…

（例）高層ビルの設計、構築の監督
・完成するための費用、時間、材料は？
・管理しやすいように、どのように部門を分けるか？
・制作された部品に互換性があるかどうかを
どうやって確認するか？
・進捗状況をどのような基準で測るか？
などなどいろんな方面から様々な質問が…

大きなソフトウェアシステムを開発する際にも、
これらと同じような広い範囲の問題に対応しなくてはならない

昔から使われている、そういう問題に対する効果的な手法があるはず。
それを使えばいいのでは？

→この推論は部分的には正しいが、ソフトウェア工学はソフトウェアの性質上、ほかの分野の工学とは根本的に異なるため、同じようにはできない。

それらの区別を認識することがソフトウェアエンジニアリング分野の発展の最初の一歩になると示された。

ソフトウェアエンジニアリング分野とそれ以外の分野との違い

他の分野（機能工学、ロボット工学など）	ソフトウェアエンジニアリング分野
・過去に設計されたパーツには比較的**互換性がある**ため、組み立て式で作ることができる。 （例）高層ビルを作る際に既製品の同じ窓の部品を使える。（例）自動車の設計者は新しいエンジンの設計は考えずとも、すでにある設計を使い、見た目だけモデルチェンジすることもできる。	・過去に設計されたソフトウェアは使える**領域が限定的である**ため、組み立て式で作ることができない（制限される。） ・そのため、**一から作る必要がある。** （例）過去のプログラムを再利用しようとしたが、特定のアプリケーションを基に設計された限定的な設計だったため、互換性がなく使用できない。
・**物理的な長さや耐久によって複雑性を定量化することができる。**そのため、比較的計画する際にかかる費用や完成品の質の予測がつきやすい。 （例）工学デバイスでは平均故障間隔（デバイスの物理的耐性）を計測することで質を定量化できる。	・**ソフトウェアの複雑さの定義は曖昧で、定量化できない。**そのため、計画する際にかかる費用や完成品の質の予測がつきにくい。 ・プログラムのクラスやファイルの多さや行数などで複雑性が評価できるのでは？ →**ソフトウェアシステムは利用される範囲が広いので、一つの視点だけでは計れない。**

	（例）ソフトウェアは工学デバイスと違って摩耗しないため、平均故障間隔で定量化することができない。

※画像はもともと Word に入っているフリー素材を使用しました。

2020/5/29

ゼミナールⅠ　レポート

[redacted]

他の分野（機能工学、ロボット工学など）	ソフトウェアエンジニアリング分野
・機械工学、電気工学などは**物理学に基づいている。**比較的厳格な地位がある。	・ソフトウェアエンジニアリングはその**ルーツを探し続けている。** ・ほかの分野と同様な厳格な地位を見つけようとしている。 →量的な方法でソフトウェアの性質を計ることの難しさが一つの要因

最近のソフトウェアエンジニアリングの研究は
実践者と理論家の二つの水準で進行している。

実践者	理論家
・すぐ使用できるような技術の研究、開発。 （例）ロボット工学、電気電子工学など	・**すぐに使用できるとは限らないが、将来的に技術に役に立つ定義、理論の研究**をしている。基礎研究。 （例）数学、物理学など
・多くの実践者によって、多くの方法が開発、推進されるため**進度が速い。** ・過去に開発した方法はだんだんと古くなるため、新しい方法との入れ替わりが激しい。	・研究は、毎年新しい発見があるわけではないが、**ゆっくりと着実に進み続けている。**

実践者と理論家の進捗に必要とされるものは膨大である。

現在、私たちの国の主要な機関
（経済、医療、政治、法執行、交通、防衛などの機関）は
大きなソフトウェアシステムによって機能している。

私たちの社会において、コンピュータシステムと、
それに関係するソフトウェアは必要不可欠である。

大きなソフトウェアシステムのエラーは重大な災害を引き起こす。

（例）防衛機関のソフトウェアシステムが、月が上昇したことによる光を核攻撃であると判断し、危うく報復攻撃を行う事件があった。（実際は人間が間違いに気づいて報復攻撃は行われなかった）

（例）ニューヨーク銀行のソフトウェアシステムにエラーが生じ、一夜にして 500 万ドル以上が損失した。

国の機能に関わるような、
大きなソフトウェアシステムの信頼性は主要な問題である。

・多くの進歩は、前もって組まれた構成要素や測定基準の欠落といった問題を克服することによって生み出される。（失敗例があるからこそ、改善点が浮き彫りになる。）

・ソフトウェア開発過程のコンピュータ工学の応用は CASE（computer-aided software engineering）を生み出した。

CASE とは何？

CASE とは CASE tool というソフトウェアを利用することで、
ソフトウェア開発過程を合理化し、簡易化することである。

CASE tools とは、コンピュータ化されたシステムの
様々な開発をするソフトウェアのこと。

CASE tool

プロジェクト計算システム	コスト推定、プロジェクト計画、 人事割り当てなどのアシスト
プロジェクトマネジメントシステム	プロジェクト開発の進捗 のモニタリングのアシスト
書類作成ツール	書類の作成、整理のアシスト
試作品の設計とシミュレーションシステム	試作品開発のアシスト
インターフェースデザインシステム	GUI 開発のアシスト
プログラミングシステム	プログラム記述やデバッグのアシスト

インターフェースとは何？

装置やユーザーとの接点、境界面である、入出力部分のこと。

（例）
・PC と周辺デバイスであれば、Bluetooth や USB、HDMI などが
インターフェースである。
・PC とユーザーであれば、キーボードやタッチパネルなどが例として挙げられ、
UI（User Interface）ともいう。

・GUI(Graphical User Interface)とはグラフィック（視覚的）の操作体系
を持つ UI のこと。

もともと一般向けに作られた word や Excel などの文書、表計算ソフトウェアや
g-mail などのメールコミュニケーションシステム
→一般の人だけでなく、エンジニアなどの開発者も使っている。

ソフトウェアエンジニアリング環境のために設計されたパッケージ。
→IDEs として知られている

IDEs とは何？

ソフトウェア開発をするためのツール（エディターやコンパイラ、デバッグなど）を一つの統合されたパッケージに結合した統合開発環境のこと。

（例）eclipse/ Visual studio/ Visual Basic など。

スマートフォンのアプリケーション開発をするシステムも IDEs の一つである。
（例）Android Studio など

プログラミングするのに必要な記述やデバッグツールだけでなく、実際にスマートフォンの画面上で書いたプログラムがどのように動作するかを見ることができるシミュレーター機能もある。

研究者、専門家、ISO を含む標準化組織の取り組みに加えて、ACM, IEEE もソフトウェアエンジニアリングの状況を改善するための競争に参加し始めている。

ISO とは何？

国際標準化機構のこと。

・計測、計算の際に使う単位（cm, mol, K, A など）や、コンセントの形状などの身近なものから、ねじの形状の企画などの専門的なものまで多くの規格を定めている。

・ISO を取得すると、製品が安全だと認めてもらえ、設計の際に使いやすくなる。
・JIS は日本の産業製品に関する規格のこと。文字コードなども含まれる。

ISO,IEEE などの取り組みは

ソフトウェア開発者の専門的技術と、個人の責任に対する無関心な態度への対応を広げる専門的な行動と倫理の規範の公認をすることから、

ソフトウェア開発組織の質を定量化するためと、それらの組織の立ち位置を改善する助けになるガイドラインを提供するところにまで範囲を広げている。

この章の残りの部分でやること。

・ソフトウェアのライフサイクルや、モジュール化のようなソフトウェアエンジニアリングの基本原則

・設計パターンの識別と適用と、再利用可能なソフトウェア構成要素の危険性
のような、ソフトウェアエンジニアリングが向かっているいくつかの方向の考察

について、論じる。

また、オブジェクト指向パラダイムが領域にあることによる影響を示す。

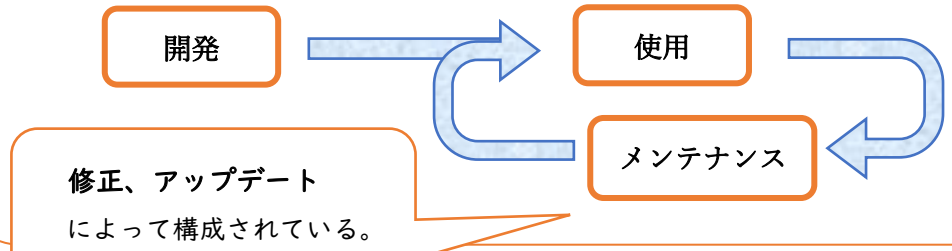
2020/5/29

ゼミナールⅠ　レポート

[redacted]

ソフトウェアライフサイクル

最も基本的なソフトウェアエンジニアリングの概念のこと。



修正、アップデート
によって構成されている。

メンテナンス段階

エラー発生

ソフトウェアの修正をすると・・・

・通信関係の変更をする必要が出てくる。
・前の修正部分にエラーが生じる。
→解決よりも問題のほうが多く発生する！！

・多くの場合、修正する人は元のコードの製作者ではないため、**まずはプログラムやそれに関する文書、少なくとも修正箇所のコードを理解するところから始まる。**

・(自分が製作者ではない)元のコードを理解し、正しく修正するのはとても難しいため、**同じ動作をするプログラムを自分で一から作る方が簡単な場合が多い。**
→これを口実にして、ソフトウェアの一部分が捨てられることはよくある。

※初期の開発段階において小さな工夫をしておくことで、
後のメンテナンス段階の際に大きな利益を出すことができる。

(例) 開発時に数値をそのままコードに書かずに定数にしておくと、修正の際に代入する数値を変更すれだけでいいため、メンテナンスの大きな手間が省ける

伝統的なソフトウェア開発ライフサイクル

要求分析

1、ソフトウェアユーザーの要望を収集し、分析する。
2、ステークホルダーと要望、費用、実現可能性について交渉し、折り合いをつける。
3、特徴を定義した要求と完成時に必要なサービスについて展開する。

・ソフトウェア開発者がいるような、企業や政府機関である場合、組織単独で実現可能性の調査を行うこともある。
・店舗で購入、またはインターネットからダウンロードできる大衆市場向けの市販品を使用する方法もある。（商用オフザシェルフ）

要求が書かれた文書のことを**要求仕様書**という。

・プロジェクトの関係者一同が同意している内容である。
・開発中に開発者と依頼者の間に意見の相違があった場合、ここに書いてある条件に沿って解決する。（最初に決めた合意内容で判断する。）
・目標は実現可能なものでなくてはならない。

→**要求仕様書**には**安定性（急な要求変更がないなど）**が必要だが、
それが保てない場合がある。

開発者と依頼者のコミュニケーション不足や要求変更は
コスト超過や**納入遅れ**を引き起こす。

ソフトウェアエンジニアはステークホルダーと、
率直で頻繁な連絡を取り合うことが必須である。

2020/5/29

ゼミナールⅠ　レポート

[redacted]

設計

要求分析	設計
提案されたソフトウェア製品の説明	ソフトウェアの構築の計画の作成
解決すべき問題を認識	問題に対する解決策
What:何をするか ⟺	How:どのように動くか

※What:何をするか、と How:どのように動くか、が逆になることはよくある。

・ソフトウェアシステムの内部構造が確立される。
・プログラムに変換できるほどの、細部にわたったソフトウェアシステムの説明が成果物である。
・**ダイヤグラム**と**モデリング**はソフトウェア設計の大事な役割を果たす。
・ソフトウェアエンジニアによって使われてきた方法論や表記法は建築分野のように安定に保つことはできない。

実装

ソフトウェア分析者（システム分析者）と**プログラマー**のタスクの違い。

ソフトウェア分析者	プログラマー
開発プロセスの全体に関わり、要求分析と設計の段階で重要になる。	主に実装段階に関わり、狭い解釈ではプログラムを書くことに対しての責任を負う。
ソフトウェア分析者と呼ばれていても本質的にプログラマーだったり、逆にプログラマーと呼ばれていてもその用語の全体的な意味ではソフトウェア分析者であったりする。 →このはっきりしない用語法はソフトウェア開発プロセスがよく混合されるという事実に見いだされる。	

テスト

昔のテスト

→デバッグ作業と、製品に要求仕様書と互換性があるかどうかを確認することと同一視されていた。

現在のテスト

→そのような見方は狭すぎる。
・ソフトウェア開発プロセスにおける中間結果や、全体のプロセスが正確にテストされるべきである。
・プログラムはただの成果物ではない。
・ソフトウェア開発の独立したステップの一つとしてみるのではなく、「要求分析と確認」、「設計と検証」、「実装とテスト」などのほかのステップにも組み込まれるべきである。

現代の品質保証をもってしても、大きなソフトウェアシステムはエラーが起き続けている。見つかることのないエラーもあれば、故障を引き起こすエラーもある。
→ソフトウェアエンジニアリングの目標の一つはそのようなエラーを排除することである。

ソフトウェアエンジニアリングの方法論

ウォーターフォールモデル

性能要求分析→設計→実装→テスト、といったように
上流過程から下流過程へ一方向に順番に進めていく開発方法。

しかし、これでは後続段階における要求仕様の変更などに対応できない。創造的問題解決にはトライ&エラーのプロセスが必要不可欠である。

インクリメンタルモデル

完成までに評価、修正を繰り返しながら完成させていく開発方法。