「ローコード・ビジネスをやって見えてきたこと」

　ローコードが世に浸透してはや数年、ローコードはその本来の威力を発揮しているのだろうか？

　ローコードツールを切り口にして、実際に開発現場が抱える根本的なジレンマについて語ります。

　ローコードを使うことが重要なのではなく、そもそものシステム開発の原点に立ち帰って「部品化」「隠蔽」の重要性について語り、さらにはそれがもたらす様々なプラス、波及効果を語りつくそうと思います。

・ローコードは銀の弾丸ではない
・世間一般の「ローコード」への誤解、SI現場の「ローコード」
・生成AIはローコードにとって代わる？

・自己紹介　　S・金子

・実務でローコードの開発やローコード活用した受託開発を展開（16年）
・一般企業に勤務
・主戦場はエンタープライズ（業務アプリ、BtoB）
・「ローコードとアジャイルはいかにして可能か」がテーマ
・土日と時間外に有志で勉強会を開催　、その他　ブログなど

・スキルセットはVB.net　SQLServer　、　JavaScriptも少々

・ローコードが開発現場を変える、銀の弾丸になる　？
　→　間違っている。そもそもローコードの本質は複雑な処理の隠蔽（カプセル化）と再利用
　「開発の民主化」がゴールではない

　→　そもそもローコード自体、「新しく」もなければ「クリエイティブ」でもない。
　やっていることは単に「複雑な処理の隠蔽」「あ、やっちまった（人為的ミス）」
　を無くすこと。

　それはライブラリの強化ということで数十年前から行われている。
　　→　　「車輪の再発明を避けよ」　と言ったような言葉で
　　→　　ライブラリの効果という原点に立ち返るべき

　（AIにも同じことが言える・・かは、まだ分からない・・・）

・「ローコードは素晴らしい」だけを謳い、ローコード　＋　手組を推奨しないベンダーは信用していない。なぜなら中規模以上の開発ではローコードで要件を完結することはできない。

　→　どのようなローコードが理想的か・・

・ローコードツールは手組の実装をアシストすべく、効果的なライブラリ群を解放すべき。
（ライブラリ群はローコードと一体のもの親和性）

そうすることでローコード　＋　手組がスタンダードになることが望ましい

・我々の考える優れたアプリケーション開発
・スパゲティ化、属人化を回避しながら、顧客の要求を受けきること
→　「優秀な人間が優秀なままに優れたプログラムを書く」　ことではない
→　優秀な人間のハイレベルな実装をパターン化してツールで代替していくということ

SI業界では「スーパーエンジニアはむしろ弊害」
⇒これが認知されていない。
・「複雑が複雑を呼ぶ」という悪循環・・複雑なコードはバグとデグレードの温床
・　二極化　　（　不幸な分断　）

・ローコード開発部隊・受託開発部隊を一体で４～５人で組成することは可能。

そうすることのメリット

・優秀なエンジニアの職人芸が「部品化」される。
その結果、「偏り」が無くなる。⇒　いい意味で「平凡化」「フラット化」
ただし、優秀なエンジニアが自分の腕の見せ所を無くすという覚悟が必要

・中級・初級エンジニアが大手を振って仕事できる
顧客から無茶な要望が来た　→　エースが腕を振るって完成　　×
→　初級・中級者が四苦八苦し、何とか稼働まで　　×
→　エースも初級・中級も最後に書き残したコードは同じ　　○

・ローコード開発と受託開発チームを一体にする大切さ
・自分達で作ったローコードの投資回収は不要　（タダ）。
・自分達で作っているので、ツールベンダーにライセンスフィー払う必要なし
・ツールのエンハンスは自分達で決める。個別案件の中からエンハンスのアイデアを得ては実装していく。

→次ページは上流工程について

・上流工程不要論者の主張がイマイチ弱い理由

⇒その３つのパターン

・スタートアップ系の若い層が「ウォーターフォール＝悪」と決めつけている

⇒　蓋を開けてみると、意欲の高いハイレベル集団、つまり
優秀なエンジニアだから「出来てあたりまえ」＝「真似できない、
参考にできない」

⇒　アジャイル・XPを妄信、　データモデルの軽視　など

・「ソースが仕様書だ！」　を地で行く高齢エンジニア

⇒　可読性の低いコード、手続き型コード（膨大）のオンパレード
「動いているものは触るな」　←　？？
退職したらただ迷惑をかけるだけ。

⇒　それでも上流工程はミニマムに抑えるべき

・プロトから受注につなげるビジネスモデルが　◎

・ローコード部分は仕様書のリバース生成が可能

・コーディング・ファースト開発の中で、可読性・変更容易性を高め、ソースを
仕様書並みに分かりやすいものに。　　常時リファクタリングモード。

・生成AIによる仕様書のリバースエンジニアリングに期待。

・普段実践として心がけていること。

・上級者の「複雑な手組」の芽を摘む。　　ーー　小刻みなコードレビュー　ーー

・「信じて信じない」　　ーーたった1行の引数を忘れるだけで大トラブル　ーー

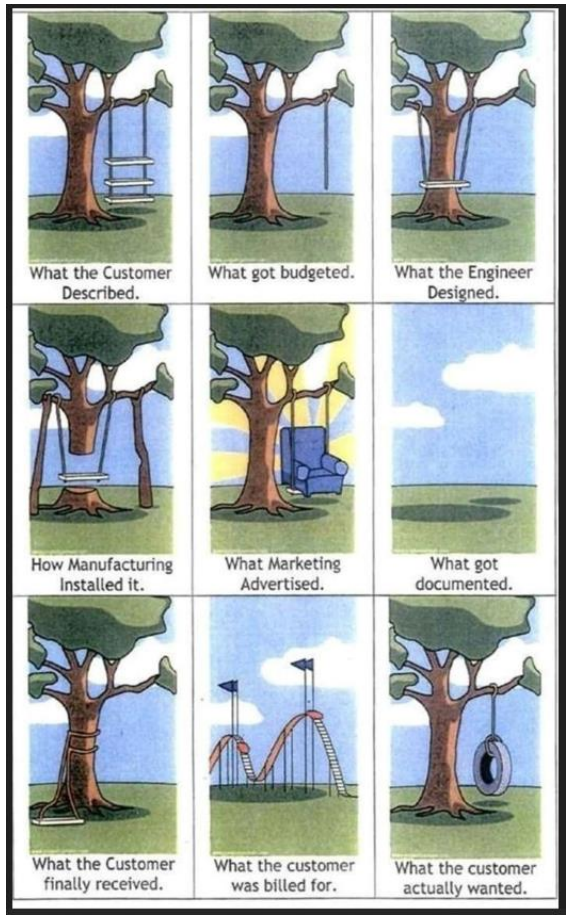
・パーサー（鋭意開発中）によるコードチェックの強化

・言語もIDEも「あれもこれもできるよ」「あとは頑張ってね」
　として、エンジニアを奈落に突き落としてきた・・・気がする
　反対に「この条件下でこのメソッドを呼んではいけない！」などと
　鬼のような老婆心で禁止・統制しまくるIDEの出現が待ち望まれる

・最近気づいたこと
　・静的解析ツールの検討をしているが、あまり当てにならない。
　　⇒　「　結果は実行時のデータや動作に依存します。　」　←　？？
　　であればＤＢ含めた動的な類推のパーサーを作る。T- SQL も対象。
　　Chat - GPTも動的類推からの指摘が優れている。

例えば、DB内のフィールド定義に依存したような処理
　If　.Item(“区分”) ＝ “A” Then（型エラー）　でも検知できるように。

40年前から繰り返されていること



ほとんどの登山ルートが「詰んでる」山

プログラミングは「怠惰」の精神が重要

怠惰なエンジニアはライブラリ化へと向かう
しかし、もし彼に「社会性」が欠如していたら・・・？

「自分用・自分たち用」のライブラリ作ってニンマリ。

「わかりやすさ　＝　民衆への解放」
「実用的・かつ汎用的　＝　どんな複雑な要求も吸収してしまう」
↑　彼にこれが作れるかどうか

「ま、いっか・・・」には2種類ある

「ま、いっか。手を抜くか・・。」
と
「ま、いっか耐えてやるか・・。」（滅私奉公）

後者がやばい。思想や一貫性を欠いた膨大なプログラム（技術負債）の放置。
DBモデルなどボタンの掛け違いをそのままにした「仕方ない」主義。
人海戦術、根性論に陥る。人月ビジネス化する。
複雑は複雑を呼ぶ、負の連鎖。

「自分は耐えれる」　←　　美しいか？後の世代に何を残す？

ご清聴ありがとうございました