ソフトウェア開発論

内部設計局面

第2.16版

- DOA/その２
- HIPO

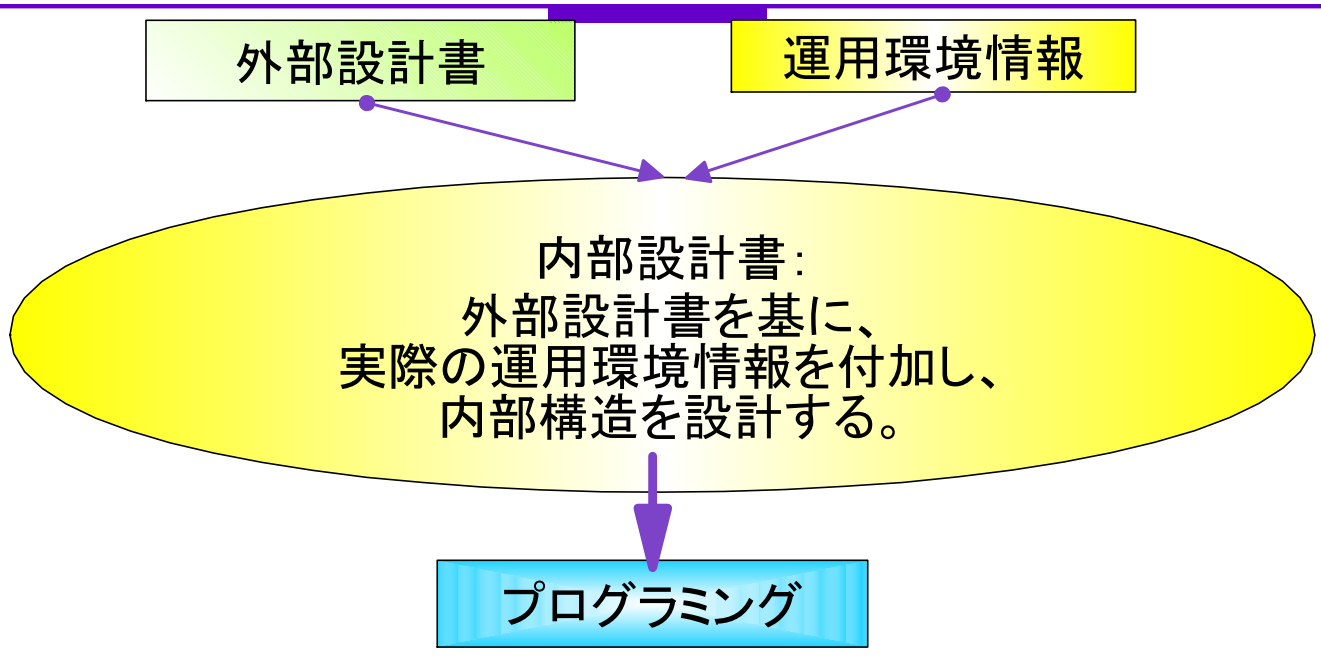
ソフトウェア開発論・スケジュール

- はじめに
- 要求分析
- 外部設計
- 内部設計局面
- プログラミング
- 単体試験
- 結合試験
- 総合試験
- 信頼性評価
- まとめ

外部設計局面／内部設計局面

- ステップ１：要求分析の結果を基に、新ソフト開発方針を固める。
- ステップ２：開発方針を基に、利用者、連携システムから見た形で、新ソフトの出来上がりを整理、設計し、キーマン、利用者に説明する（外部設計書作成）。
- ステップ３：外部設計を基に、実際の運用環境情報を付加し、内部構造を設計する。（内部設計書作成）

ステップ３：内部設計書作成



運用環境情報とは？

- 開発対象ソフトウェアの限界、弱さ(BQC)を知りたい。
  そのために、稼働時の運用情報、環境情報を明確にし、それから来る例外事項を疑問に浮かべること(if then分析)をおこない、thenが発生したときの影響を調べる。
- 運用情報とは: これから新たに開発するソフトウェアが、運用時に実際に載るコンピュータ(サーバーやクライアントのこと)
  - コンピュータの精度
  - データベースの特性(並列処理が不得意)
  - コンピュータの容量や速度
  - ・・・
- 環境情報とは: 運用時に、上のコンピュータを取り巻く環境 (想定外の利用も考慮)
  - 各種の利用者
  - ネットワーク
  - 連携システム
  - 建物などの設備
  - 各地のカントリー・リスク
  - ・・・

運用情報から来るBQC

- コンピュータの精度
  - 返済シミュレーションでは、日歩計算で浮動小数点演算が必要
    - どの桁で？、四捨五入？切り上げ？切り捨て？
  - 有効数字桁数にも注意:32ビットマシンの単精度・浮動小数点演算は何桁まで可能？
- データベースの特性
  - 関係データベースモデルでは、データの一意性を重視する。 そのためにデータベースは集中処理になる。 そうすると、多くの人からの処理要求が集中し、処理効率が落ちる。 それを避けるために、内部設計書では順序性を明確に指定する。
  - 例:
    - 外部設計書:**当日、それまでの、料理名ごとの提供数を答える。**
    - 内部設計書:注文情報表、もしくは提供料理情報表が更新中の場合は、**更新終了を待つ**。 注文情報表および提供料理情報表が他の人に利用されないように**ロックを掛ける**。 その後、注文情報表の本日分を呼び出し、料理名ごとの提供数を提供料理情報表に書き込む。 その時に、料理名はソートして書き込む。

データベースは直列動作

環境情報から来るBQC

- 環境から来るBQCの見つけ方

このスキルが、SE適性を決める

- **(想定外を含む)利用者ごとに**→動作およびその頻度
  (後で内部設計、結合試験で利用)
  →その動作をしくじったら、あるいは誤解して操作したら
  (例外事項への疑問)どうなる?(if then分析)
  →BQC (開発対象ソフトの限界、弱さ) が見つかる。
  →それへの解決するかしないか。する場合には、保険をかけるか、人間系で回避するか、設計に盛り込むか、別のシステムでやるかを決める。

環境情報からくるBQC
例:注文管理システム設計のBQC発見

WHO	WHEN (CONDITION)	WHAT(ACTION)	例外事項への疑問 (if then分析)	BQC(内部設計の限界、弱さ)を捜す	対策
接客係	一日40組 (最大同時4組)/二店舗	・お客さま: 料理を注文する。 ・接客係: 料理名数と売り上げを来客組ごとに記録する。 ・私: 領収書を発行する。	・お客さまが注文を取り消したらどうなるか?	・提供料理情報との整合性がとれなくなる。	提供料理情報を再計算する。[内部設計変更で]
			・材料切れのため、料理を出せないとどうなるか?	・注文情報変更が必要	上と同じ対策が必要
			・紙切れのため、領収書を出せないとどうなるか?	・手書きで領収書を発行する事になっている。	設計変更なし [なにもしない]
			・サーバ停電の場合はどうなるか?	・30分以内は大丈夫/それ以上はシステム停止	手でサーバ停止し、手作業で代行する。[人間]
			・各店のPCが故障したらどうなるか?	・手作業で業務実施。後で入力が必要	各支店PCを2台とする。[保険で対処]
材料発注システム	一日一回	・材料発注システム: 問い合わせをかける。 ・私: 当日、それまでの、料理名ごとの提供数を答える。	・問い合わせがなかったらどうなるか?	・誰も気がつかない。	店長が退社時にチェック
			・接客中に問い合わせがあるとどうなるか?	・大丈夫な設計になっている。	設計変更無し
			・一日二回問い合わせがあるとどうなるか?	・そのたびに答えることになる。	それでよいか、材料発注システムを点検[他システムで対処]

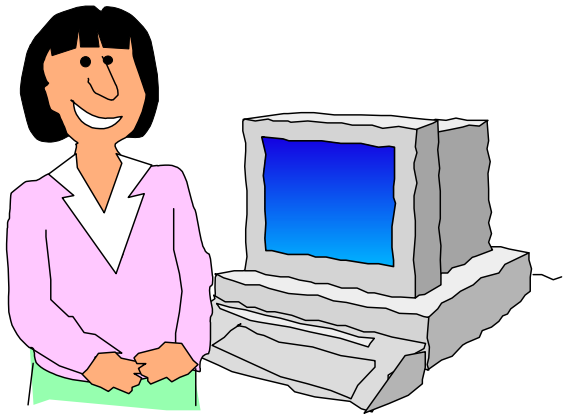
- 運用情報・環境情報
  - 運用時間:朝10時から夜8時(営業時間内)、土曜日曜も稼働
  - サーバ:本店に設置。予備機はなし、無停止電源(30分)付き
  - 窓口PC:各支店に1台配置、32ビット演算機
  - 接客係:各支店　接客係人数+予備2台配置

対策とるならフェイル・セーフで

- フェイル・セーフとは： 不測の事態になったときに、より安全側の設計を選択すること。
- 例： 水道の蛇口は押さえると止まるように、関西大震災のあとで統一された。
  - 設計としては押さえると止まる、押さえると出る　二通りが可能である。
  - 製造コストはどちらも変わらない。
  - が、物が落ちてきたときに、水が止まる設計のほうが、より安全である。
- 例： 信号システム
  - コストの変わらない二つの設計、どちらを選択するか？
    - 青ならば赤に変える。　　if (信号=='青') 信号='赤'
    - 赤でなければ赤に変える。　if (信号<>'赤') 信号='赤'

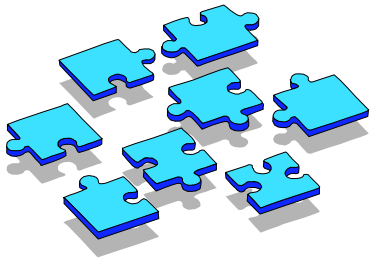
演習：環境情報からくるBQC

- 返済シミュレーション・ソフトについてBQCを見つけて見ましょう。
  - まず、運用情報を決める。
  - 決めた運用情報(制約)のもとで、(想定外)利用者群の追加、およびそれを取り巻く環境情報を考える。
  - 上の環境情報からくる例外事項への疑問を出し、BQCを探し、対策を考える。



内部設計書とは？

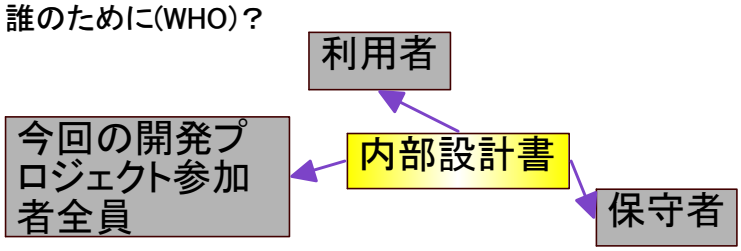
注文管理システム設計
－データ分析
－データ制約

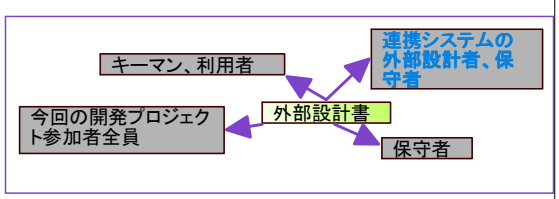


外部設計書　内部設計書　実物

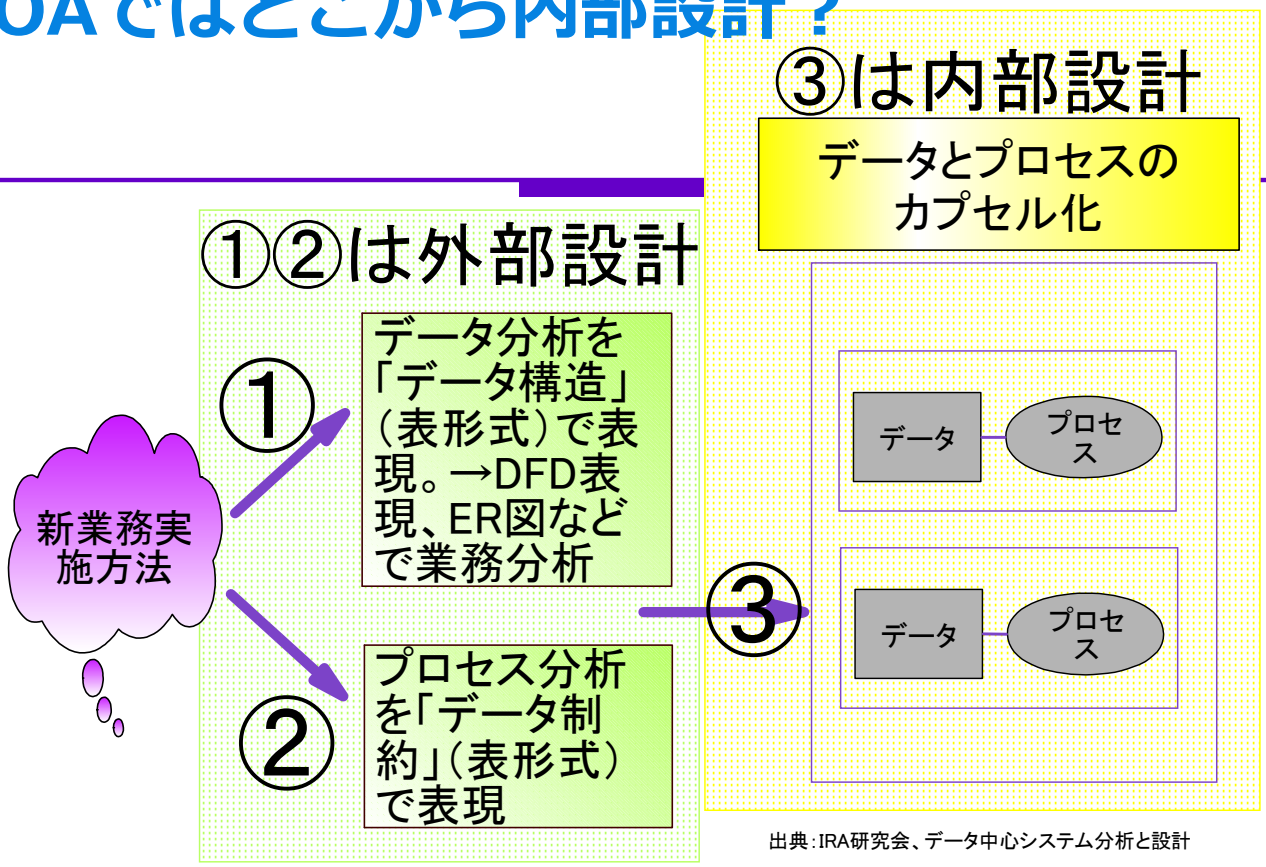
内部設計書とは？

- 何を作るの(WHAT)？
  - 外部設計書を実現するシステムを製造するために、**プログラム部品（データとプロセスのカプセル）の構造**を設計する。 背景にある考え方：ソフト開発は、プログラム部品を組み合わせて、糊付けをするという発想で実施する。
  - 運用環境情報からくる、「**限界**」を明示する。
    言い換えれば、エラー処理及び人間が運用でカバーする部分を明記する。
  - 順序性を分析し、**直列性**を持たせるように設計する。
- 何のために(WHY)？
  - プログラマー、結合試験者、単体試験者に出来あがる内部設計を説明し、
    - 複数の人間が平行して作業できるようにする。
  - プログラム、単体試験テストケース、プログラム開発体制、単体試験体制は、内部設計書に基づき作成する。
  - 利用者と、運用でカバーをする部分について協議を行う。
- 誰のために(WHO)？





DOAではどこから内部設計？



出典：IRA研究会、データ中心システム分析と設計

これが重要

昨日の外部設計おさらい
注文管理システム－データ分析

私（注文管理システム）からみて、

- 私と対話する（よってくる）人、システムは誰？
- 人、システムごとに、どんなときに、どんな頻度でやってくる？
- 対話はどんなデータを使って、どんなプロトコル（操作手順）でやられるか？
- 対話に使われるデータの詳細は何？

ソフト開発に必要な、外との約束データ

操作は上から下へ

WHO	WHEN(CONDITION)	WHAT(ACTION)	WHAT(DATA)
接客係	一日40組（最大同時4組）／二店舗	・お客さま：　料理を注文する。 ・接客係：　料理名数と売り上げを来客組ごとに記録する。 ・私：　領収書を発行する。	注文情報｛ 来客組ごとに、提供した料理名、数と、それによる売り上げ｝
材料発注システム	一日一回	・材料発注システム：　問い合わせをかける。 ・私：　当日、それまでの、料理名ごとの提供数を答える。	提供料理数情報 ｛当日、料理名ごとに、提供数｝

昨日の外部設計おさらい
注文管理システム設計－データ制約

表	属性	ドメイン制約	主キー	存在制約	参照制約	多重度制約	導出制約	関連制約	更新制約	処理順序制約
A= 注文情報	日付	{yyyy/mm/dd}、年月日	YES	必須	なし	1	なし	なし	なし	なし
	来客組ID	{hh:mm}、時刻+ { 001..999 }、整数	YES	必須	なし	1	なし	なし	なし	来客順に振る。
	C=料理名	{ 2..20 }、文字	YES	必須	F	1	なし	なし	なし	なし
	D=注文数	{ 1..99 }、整数	NO	必須	非該当	1	なし	Cによる制限	なし	なし
	E=売り上げ	{0..999999 }、整数	NO	必須	非該当	1	なし	なし	D<=0 E=0	支払い時確定
B=提供 料理数情報	F=料理名	{ 2..20 }、文字	YES	必須	なし	1	なし	なし	なし	名前順にソート
	提供数	{ 1..999 }、整数	NO	有効な主キー、C,D存在	非該当	1	当日C別D合計	なし	なし	A更新時以外に計算

参考： データ分析の結果

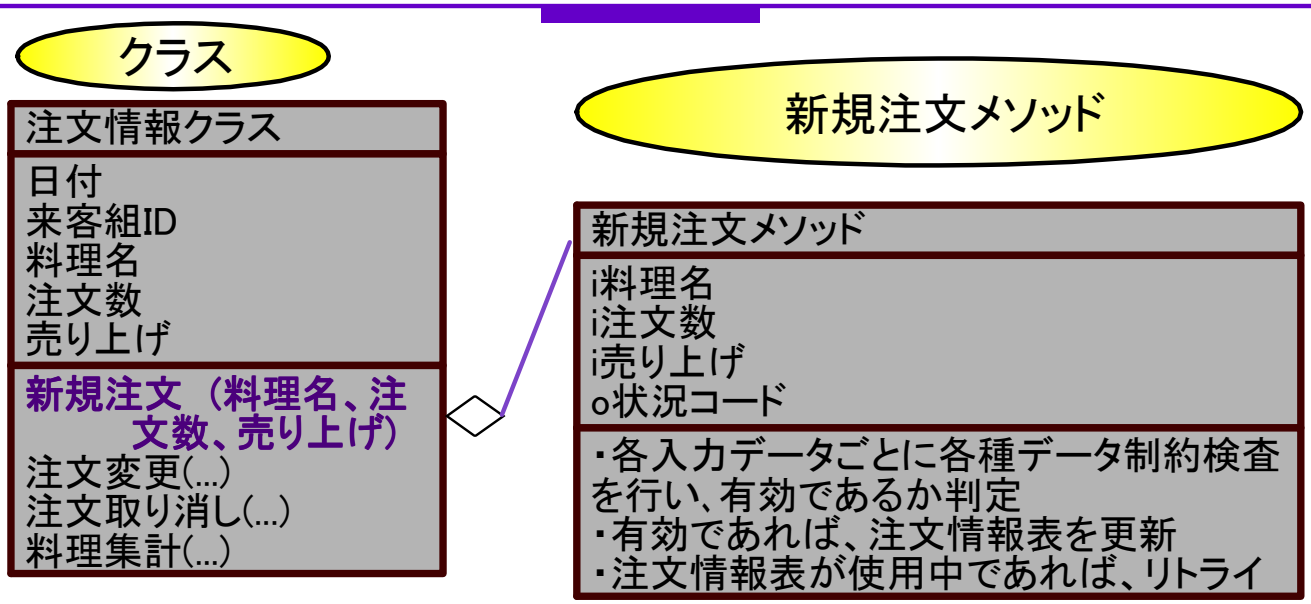
WHO	WHEN(CONDITION)	WHAT(ACTION)	WHAT(DATA)
接客係	一日40組（最大同時4組)/二店舗	・お客さま： 料理を注文する。 ・接客係： 料理名数と売り上げを来客組ごとに記録する。 ・私： 領収書を発行する。	・注文情報｛ 来客組ごとに、提供した料理名、数と、それによる売り上げ}
材料発注システム	一日一回	・材料発注システム： 問い合わせをかける。 ・私： 当日、それまでの、料理名ごとの提供数を答える。	提供料理数情報 {当日、料理名ごとに、提供数}

木馬亭・注文管理システムのクラス構成

MVCモデル= Model + Controller + View

参考：http://ja.wikipedia.org/wiki/Model_View_Controller

クラス＝データ＋プロセス　のカプセル化（UML表現）



カプセル化：　PCから注文を出すには、
必ずサーバ側新規注文メソッドを使うことになる。

メソッドの設計－HIPO

新規注文メソッドの機能：
- ・データ制約に該当しているか検査
- ・問題がなければ、注文情報表を更新
- ・注文情報表が使用中であれば、リトライ

入力データ	プロセス	出力データ
■料理名 ■注文数 ■売り上げ	■状況コード←'正常' ■IF 料理名の桁数が{　2..20文字　}でなければ、状況コード←'E001' ■IF 料理名が'特上サーロインステーキ'　ならば　注文数が{　1..4　}以外は状況コード←'E002' ■ELSE　IF 注文数が{　1..99　}以外ならば状況コード←'E003' ■IF　売り上げが{0..999999　}以外ならば状況コード←'E004' ■IF　状況コードが'正常'ならば ■　新規注文表へ日付、来客組ID、料理名、注文数、売り上げを挿入 ■　IF　新規注文表.sqlcodeが0でなければ、状況コード←'E005' ■　IF　新規注文表が使用中ならば' ■　　REPEAT　0．5秒待つ。 ■　　新規注文表へ日付、来客組ID、料理名、注文数、売り上げを挿入 ■　　IF　新規注文表.sqlcodeが0でなければ、状況コード←'E005' ■　　IF　新規注文表が使用中以外ならば、REPEATを抜ける。 ■　　END-REPEAT ■　END-IF ■END-IF	■状況コード

プロセスの記述法：
- ■順序性の規則がある。同じレベルの文章では、上から下へ。文章中では左から右へ。
- ■構造を明らかにするために、字下げおよび、制御キーワードをつけて記述する。

A社データ制約とカプセル化

データ制約の種類	ドメイン制約	識別子（主キー）	存在制約	参照制約	多重度制約	導出制約	関連制約	更新制約	処理順序制約
A=顧客ID	{1000000..9100000}、整数	YES。顧客DBを通してユニークであること	必須	参照不可	1	なし	なし	なし	見積もり後、顧客が望む場合は与信システムに処理を引き継ぐ
B=本日の金利	{<0%..23%}、整数もしくは実数	NO	必須	非該当	1	なし	なし	なし	なし
C=返済予定日数	{0..3650}、整数	NO	必須	非該当	1	なし	なし	なし	なし
E=返済額	{0.8桁}、正の整数	NO	有効なA,B,C存在	非該当	1	E=(1+B/36500)**C 1円未満は切り上げ	なし	なし	なし

導出制約の結果だけ、
PC側返済シミュレーションクラスに見せる。

返済シミュレーション・ソフト
内部設計例

顧客DBへのアクセスは
（**返済額計算メソッド**経由のみと）
厳しく制限されている！

(サーバー側)顧客DBクラス
- i顧客ID
- oお客さまのお名前
- o元金
- o(電話番号)
- o(住所)

・元金取得メソッド
・返済額計算メソッド

(クライアント側)
返済シミュレーションクラス
- iA=顧客ID
- iB=本日の金利
- iC=返済予定日数
- oお客さまのお名前
- oE=返済額
- o元金

・A,B,C入力
・A,B,C ドメイン制約検査
・**返済額計算メソッド呼び出し**
・印刷

n

1

(サーバー側)返済額計算メソッド
- iA=顧客ID
- iB=本日の金利
- iC=返済予定日数
- oお客さまのお名前
- oE=返済額
- oD=元金

・元金取得メソッド呼び出し、元金情報入手
・A,B,C,Dは有効か検査
・B,C,DよりEを計算
・Eに対してドメイン制約検査

多部門から共同利用される、顧客DB

演習：返済シミュレーション・ソフト　メソッド・レビュー

HIPO（内部設計）

メソッドの機能：返済額計算メソッド		
入力データ	プロセス	出力データ
■ A：顧客ID ■ B:本日の金利 ■ C:返済予定日数	■ 元金取得メソッド呼び出し （入力：顧客ID、出力：お客さまのお名前、元金） ■ IF 不明の顧客IDであれば、エラー　//顧客ID有効か検査 ■ IF 元金＞0でなければ、エラー　　//元金は有効か検査 ■ 本日の金利、返済予定日数、元金より、返済額を計算 ■ 返済額に対してドメイン制約検査	■ お客さまのお名前 ■ D:元金 ■ E:返済額

A社データ制約（外部設計）

データ制約の種類	ドメイン制約	識別子（主キー）	存在制約	参照制約	多重度制約	導出制約	関連制約	更新制約	処理順序制約
E=返済額	{ 0..8桁 }、正の整数	NO	有効なA,B,C存在	非該当	1	E=(1+B/36500)**C 1円未満は切り上げ	なし	なし	なし

運用環境情報

返済額の計算は、単精度は32ビットのPCで実行、
浮動小数点演算だと単精度では有効数字7桁半

おわり