ライフサイクル境界に着目した 対話型業務アプリケーションの設計指針

井田明男† 金田重郎† 矢野寛将†
Akio Ida† Shigeo Kaneda† Hiromasa Yano†
†同志社大学大学院・理工学研究科
† Graduate School of Science and Engineering, Doshisha University

要旨
業務アプリケーションがオブジェクト指向アプローチで設計されるようになって久しい．それらの多くは，基本的にはヤコブソンの OOSE に端を発する手法に則って設計されてきた．それらは概してユースケース駆動であり，ユースケースの論理的実現のために界面（boundary），制御（control），実体（entity）の３種類のクラス役割を導入する設計手法である．これらの役割の導入は，要求仕様変更の影響を受けやすい部分とそうでない部分を分離する．しかしながら，control の導入は本質的に手続き指向であり，カプセル化された部品による自律分散制御を目指すオブジェクト指向から逸脱してしまう問題が残されている．そこで，本稿では，従来のアプローチに実体間の存在従属性に着目したグラフを導入し，実体オブジェクトのライフサイクル境界に仮想エンティティを配置する設計指針を提案する．また，設計品質を定量的に評価する指標（ドメインモデル貧血度，および制御役クラスの依存度）を併せて提案する．提案手法を典型的な対話型業務アプリケーションの設計に適用した結果，設計上の迷いが解消し，これらの指標値も改善したため，提案手法が設計品質の向上に効果的であると判断する．

1. はじめに

業務アプリケーションがオブジェクト指向アプローチで設計されるようになって久しい.それらは ICONIX[1]にせよ RUP[2]にせよ，基本的にはヤコブソンの OOSE[3]に端を発する手法に則って設計されてきた．それらは概してユースケース駆動であり，その論理的な実現のためにクラス役割として界面（boundary），制御（control），実体（entity）を導入した設計手法（以下，BCE アプローチと記す）である．

これらの設計手法は，仕様変更の影響を受けやすい部分とそうでない部分，そして，問題のレイヤと実現手段のレイヤを明確に分離する優れた設計手法である．しかしながら，界面クラス，制御クラス，実体クラスへ責務をどのように配分するかについてのガイドラインは多様であるため，設計者によって設計成果物の品質にバラツキが生じ，ともすれば，ドメインモデル貧血症と呼ばれるアンチパターンに陥りやすい問題が残されている．なぜならば，制御役のクラスを導入した途端にその設計をオブジェクト指向から手続き指向へと変質させてしまうための選択肢が用意されたことになるからである．

そこで，本稿は対話型の業務アプリケーションを設計対象として，従来の BCE アプローチに存在従属性の概念を導入することを提案する．具体的には，集中制御を排除し，仮想実体を導入する．提案手法によってドメインモデル貧血症を回避するとともに，より工学的に設計構造を導くことを目指す．以下，第 2 章では，本提案に関係する概念の定義と関連研究の概要を述べる．第 3 章では，本提案の手法を説明する．第 4 章では，成果物を指標を用いて比較することによって提案手法の効果を検証する．第 5 章は，まとめである．

2. 関連する概念と先行研究

2.1. 存在従属性

存在従属性の概念は P．チェンが紹介した[4]．あるオブジェクトが，別のオブジェクトの先立つ存在を前提として存在し得るとき，前者のオブジェクトは後者のオブジェクトに存在従属するという．そして前者のオブジェクトを弱実体（weak entity type）のオブジェクト，後者のオブジェクトを強実体（strong entity type）のオブジェクトと呼ぶ[4]．存在従属性を導入することよって，たとえば，ある注文は，注文主である顧客と注文対象である商品に存在従属する．月々のローンの返済は，過去の購買という事象に存在従属する，などといった表現が可能となる．しかしながら，なぜかこの概念はオブジェクト指向モデリングの教科書で紹介されない．UML の提案者たちの OOA の文献ではわずかに[5]だけが存在従属性について言及している．

2.2. 存在従属性と属性

存在従属性は，オブジェクト間だけの関係ではない．オブジェクトとその属性の間にも存在従属性が認められねばならない．すなわち，属性とは，あるオブジェクトが生まれるとともにそのオブジェクトの値として生まれ，そのオブジェクトがこの世に存在する限り，そのオブジェクトに付随し，そのオブジェクトがこの世から消えるとともに当該属性も消えるデータ項目のことである．したがって，属性群はオブジェクトに存在従属

するといえる．言い換えれば，「オブジェクトに存在従属するデータ項目こそが属性」という観点でそれらを定義すると，属性はオブジェクトに完全関数従属するとともに，候補キーではない属性から候補キーを構成する属性への関数従属性をも排除することができる．クラスが持つべき性質で必要なのは，「すべての事実は，キーだけに関係する事実である[6]」を満たすため，これだけでも少なくともボイス・コッド正規形が達成される．

しかしながら，佐藤[7]は，あるオブジェクトについてそもそも何が属性として帰属するのか，という判断は，「本質的属性」という（哲学的に）厄介な論点－存在論とか形而上学など－を考慮しなければならない，とする．そのため，筆者らは，「業務要件に照らして」帰属すると見做せるならば，属性と考えてよいことにしている．たとえば，社員の属性として入社年月日は本質的属性ではないにせよ，実用上は属性と見做す．

筆者らは，先行研究において，存在従属性に着目してオブジェクト指向分析を実施するとボイス・コッド正規形以上の頑健さを備えたドメインモデルが構築できることを明らかにしている[8]．存在従属性の概念は理解しやすくドメイン中で見極めやすいため，提案手法により，第 4 正規形以上の正規化レベルを持ったドメインモデルが容易に構築できる．実装時には RDBMS を単なる永続化層としてではなく，データの整合性を保つための砦として使用できることに繋がる．

2.3. 存在従属グラフ

同時に筆者らは，ドメインモデルを存在従属グラフで表記することを提案した[8]．これはビジネスドメインにおける実体間の存在従属性を表現するためにクラス図の表記を流用した静的構造図である．そして，存在従属グラフは，ビジネスドメインにおける実体のインスタンスの存在の因果関係（時間的な存在の半順序関係）を表現するため，動的モデルでもある．

以降の説明では，「会員が商品を注文する」という対話型業務アプリケーションの典型的なユースケースにおける最小限のモデルを例題として用いる．販売管理者が会員が発した注文を捕捉・管理するドメインを想定している．図 1 はそのようなドメインの存在従属グラフの例である．図 1 は省略可能な要素をすべて省略した表記であるが，文献[8]のガイドラインによって図 1 と図 2 は等価である．

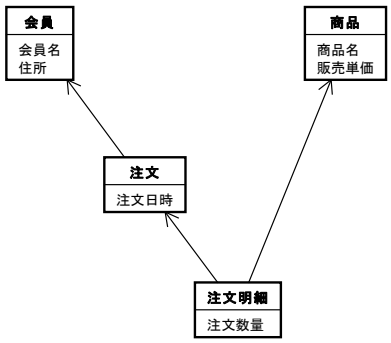

図 1：省略可能要素をすべて省略した存在従属グラフの例

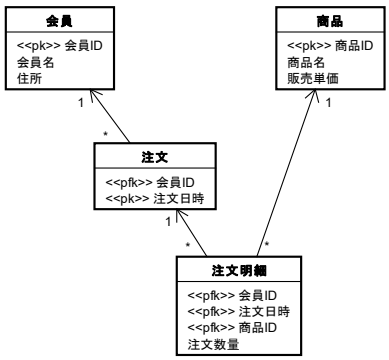

図 2：詳細に表記した存在従属グラフの例である．図中の《pk》は主キー，《pfk》は主キー兼外部キー

2.4. ドメインモデル貧血症

ドメインモデル貧血症[9]は，現存する多くの対話型業務アプリケーションに見られる症状である．それらの設計を見ると，概して実体クラスの振る舞いの責務が極端に薄く，その操作のほとんどはそれらの属性へのアクセッサである（図 3）．反面，制御役のクラスの責務が大きく，さまざまなクラスと依存関係を持つタコ足クラスになっている（図 4）．これはオブジェクト指向とは真逆の設計であり，制御役のクラスは再利用性，保守性ともに芳しくなくカプセル化のメリットをほどんど活かせていない．

3. 提案手法

提案手法では，存在従属グラフに登場する実体間の存在従属関係から自動的に導かれることはすべて設計情報として利用する．そして，クラスやテーブルの定義は自動生成を前提としている．以下では，引き続き受注管理の例題を用いながら，提案手法のプロセスとガイドラインを説明する．

3.1. 【STEP1】存在従属グラフを作成する

要求記述から，記述に登場する概念間の存在従属性に着目して，存在従属グラフを作成する．作成手順は文献[8]に従う．そして，この例題の存在従属グラフは先述の 2.3 節の図 1，図 2 の通りである．

3.2.【STEP2】存在期間が等しい実体をグループにまとめる

つぎに，存在従属グラフに登場する実体をそれらの存在期間に着目してグループ分けを行う．そして，各グループには概念としての名前を命名する．図 5 はグループ分けを行った結果を示している．

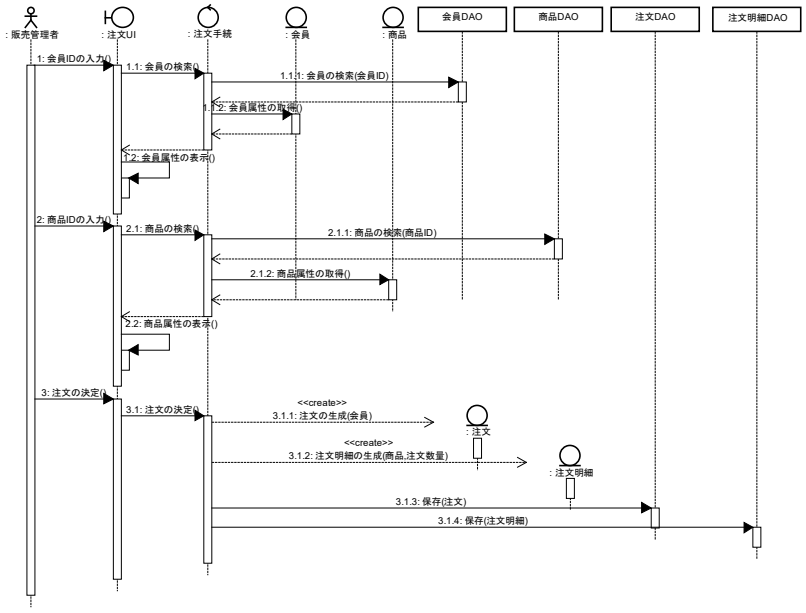

図 3：ドメインモデル貧血症の症例：実体クラスの責務が薄い[1]

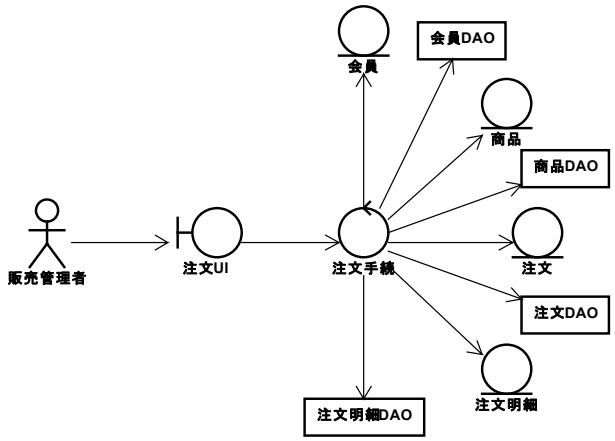

図 4：ドメインモデル貧血症の症例：タコ足の制御役クラス

3.3.【STEP3】グループ毎にそれぞれを管理するユースケースを生成する

筆者らの先行研究[10]は，存在従属グラフからそれを操作するユースケース群が導けることを明らかにした．例題では，「商品を管理する」，「会員を管理する」，「注文を管理する」の 3 つのユースケースが導かれる（図 6）．なお，「管理する」は，当該オブジェクトの新規追加，一覧の照会，更新，削除の機能の必要性を含意する．

導出された 3 つのユースケースのうち前者の 2 つは，いわゆるマスタ・メンテナンスのためのアプリケーションに相当する．それらは，対象実体の単一あるいは複数のインスタンスを処理対象とするが，いずれも 1 つのエンティティのインスタンスであるため，アプリケーションの設計はシンプルである．

[1] DAO は Data Access Object の略であり，実体のインスタンスを不揮発性の記憶域へと永続化したり，永続化されている実体のインスタンスを主記憶域に復元するためのインターフェースである．

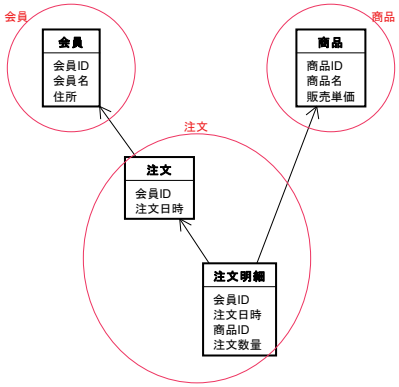

図 5：存在期間で実体のグループ分けした存在従属グラフの例

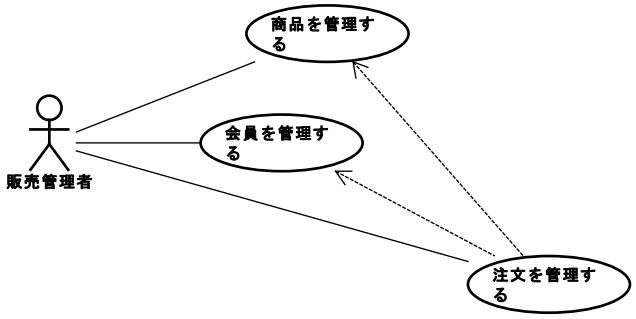

図 6：存在従属グラフから導かれるユースケース

しかしながら，3 つ目ユースケース「注文を管理する」に関しては，会員，注文，注文明細，そして商品という 4 つ実体に跨る処理を行う必要があるため，設計は複雑になる．複雑さの原因は，ライフサイクルが異なる実体が分離されていることである．たとえば，今舞い込んだ注文を登録するために，過去に登録済みの会員と商品の属性をそれぞれ保護した上で参照しなければならない．

3.4.【STEP4】ユースケース記述としてのイベントフローの生成

おのおののユースケースについて，使用性を考慮したユースケース記述としてのイベントフローを生成する．生成時は**図 7** の対話型業務アプリケーションの典型的な画面遷移を適用する．図 7 は筆者らの経験から得たものである．生成イベントフローは次の通り：

アクターは，メニュー画面から実施すべき業務を選択すると，当該業務で扱う実体のインスタンスの一覧が表示される．もしも一覧の件数が多すぎる場合は，一覧画面にて絞込条件を指定することもできる．一覧から 1 つのインスタンスを選択すると[2]，詳細画面に遷移し，そこで，属性値を編集したり，あるいはインスタンスを削除することができる．なお，削除する場合は，確認画面が表示される．

3.5.【STEP5】仮想実体を導入する

生成イベントフローを実現するために，仮想実体を導入する．その位置は，存在期間が異なる実体グループの間である．この目的は，存在期間が異なる実体をあたかも 1 つの実体であるかの如く，より上層のレイヤに対して見せることである．仮想実体そのものはインスタンス化されず，永続化の対象ともならない．丁度 RDBMS における仮想表に対応する概念である．3.3 節では，単一の実体だけを扱う業務アプリケーションの設計はシンプルであると述べたが，複数の実体を跨ぐ操作を行うアプリケーションの場合であっても，仮想実体を導入することによって，処理対象をあたかも 1 つの実体に見せることができれば，その設計は同様にシンプルにできる．なお，仮想実体の命名は，グループを代表する実体名を連結したものを採用した．

例題の場合は，ライフサイクルが異なる会員と注文グループ，注文グループと商品の間にそれぞれ仮想実体を定義する．図 8 は，仮想実体を定義した状態の存在従属グラフである．会員と注文グループの間には「注文会員」が，注文グループと商品の間には「注文商品」をそれぞれ定義している．さらに，「注文会員」と「注文商品」の間に依存関係があるため，アプリケーションから見た実体はあたかも 1 つにみえる．この場合は，「注文会員」だけを上位レイヤに提示することになる．

また，仮想実体の責務も存在従属グラフから自動的に導ける[10]．それらの知識の責務は，仮想実体が跨ぐ 2 つの実体が保持する属性の和集合である．また，振る舞いの責務は，主に独立クラス側のインスタンスの検索，依存クラス側のインスタンスの生成，そして，両者の属性値の管理である．属性の管理には，小計や合計の計算などの小さなビジネスロジックも含まれる．また，独立クラス側のインスタンスの属性値を保護することも大切な責務である．

[2] 新規登録の場合は，一覧上の新規インスタンスを選択する

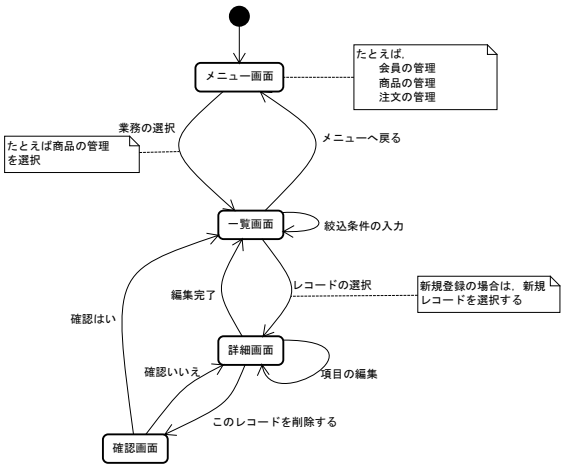

図 7：経験から得られたユーザフレンドリな UI 画面の遷移とイベント

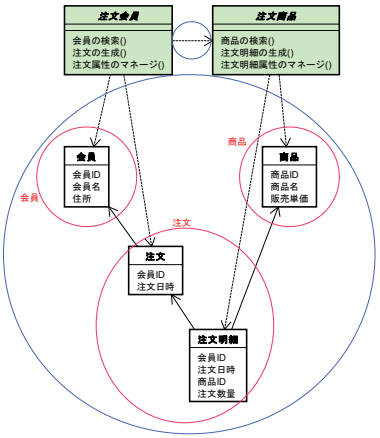

図 8：生存期間が異なるグループ間に仮想実体を定義した例

例題の場合は，「注文会員」の知識の責務は，会員 ID，注文日時，会員名，住所であり，「注文商品」のそれは，会員 ID，注文日時，商品 ID，注文数量，販売単価である．また，「注文会員」の振舞いの責務は，注文を発した会員の ID に存在従属する属性群（会員名，住所）を同定すること，新規注文のインスタンスを生成し，生成時点を注文の注文日時に設定すること，そして，注文の識別子である会員 ID+注文日時と，仮想実体「注文商品」の識別子の一部である，会員 ID+注文日時を使って，注文会員と注文商品の 1 対多の関係構築することである．具体的には，「注文商品」の識別子の上位 2 つの項目に当該注文の会員 ID と注文日時を転記する準備を行う．そして，界面役のクラスから商品 ID と注文数量が指定される都度，先程準備した ID と併せて注文商品に注文明細インスタンスの生成を委譲する．

仮想実体「注文商品」の責務は，「会員注文」から注文明細の生成を委譲される都度，指定された商品 ID の属性群（商品名，販売単価）を同定する．そして注文明細のインスタンスを生成し，指定された注文数量と販売単価を掛け合わせて金額を計算する．

3.6.【STEP6】界面，仮想実体，実体，永続化界面（DAO）の相互作用を検討する

STEP4 で作成したイベントフローをもとに，ユースケース実現の相互作用を検討する．すでに実体と仮想実体の責務が導かれているため，相互作用の検討がストレートフォワードな作業となる．図 9 は例題について，相互作用を検討した結果のシーケンス図である．制御役は排除され，仮想実体も含めて，実体の責務が豊かになっている．また，図 10 は，提案手法におけるユースケースのクラス図である．責務が存在従属グラフの構造にしたがって偏りなく配分されている．

4. 設計成果物の比較

本章では，提案手法を適用前と適用後の成果物を比較する．そのためにドメインモデルの貧血度，および制御役クラスの依存度（タコ足度）という指標を提案して用いることとする．

ドメインモデル貧血度は，実体クラスの操作に占めるアクセッサの割合であり，適用前は 100%であったのに対して，適用後は，25%に減少している．制御役クラスの依存度は，制御役の依存先クラス数を制御役クラスの操作数で割ったもので，適用前は 2.6 であったのに対して，適用後は，1.6 に減少している．このことから提案手法は適切な設計に効果ありと判断する．

5. まとめ

本稿では，対話型業務アプリケーションが扱う実体間の存在従属性に着目した設計戦略を提案した．併せて，設計成果物の品質を定量的に評価するための指標も少ないながら提案した．ファウラー[11]，マーチン[12]，エヴァンス[13]，ヴァーノン[14]などの文献にみられるように，ドメインモデル貧血症対策も含めて，オブジェクトへの適切な責務配分のための設計指針を提案している先行研究は数多い．しかしながら，筆者らは設計指針は少ない原則で構成され，簡単に学べて，強力なものでなければならないと考える．本提案は存在従属性をオブジェクト指向アプローチおける設計戦略として位置づけたところにオリジナリティがある．提案手法がオブジェクト指向開発における設計の迷いを少しでも解消し，工学的な設計のための一助となれば幸いである．

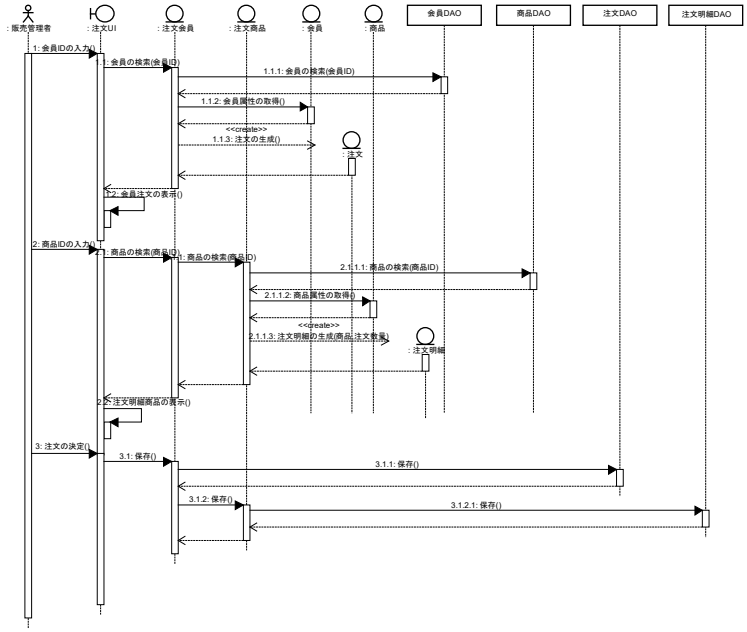

図 9：提案手法のシーケンス図：実体クラスの責務が豊かになっている

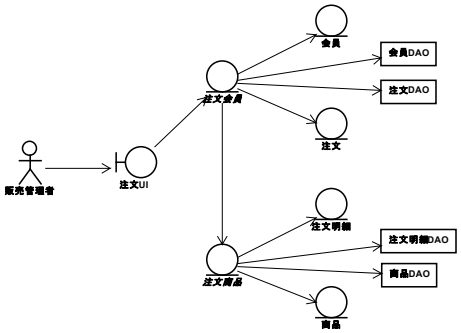

図 10：提案手法のユースケースのクラス図：タコ足クラスが解消されている

参考文献

[1] ダグ・ローゼンバーグ：ユースケース駆動開発実践ガイド，翔泳社，2007

[2] フィリップ クルーシュテン：ラショナル統一プロセス入門第 2 版，ピアソンエデュケーション，2001

[3] I. ヤコブソン他：オブジェクト指向ソフトウェア工学 OOSE―use - case によるアプローチ，トッパン，2015

[4] P. Chen：The Entity Relationship Approach to logical Database Design，QED information science, Wellesley, 1977

[5] James Rumbaugh：オブジェクト指向方法論 OMT―モデル化と設計，トッパン，1992

[6] C．J. Date：Database In Depth，Oreilly & Associates Inc, 2005

[7] 佐藤正美：データベース設計論―T 字形 ER，ソフト・リサーチ・センター，2005

[8] 井田明男，金田重郎，熊谷聡志，藤本明莉：存在従属性に着目した論理要件ロバストなドメインモデルの作成―ドメインクラス図をユビキタス言語として用いるために，情報処理学会・論文誌，5 月，2015 年

[9] Martin Fowler： Anemic Domain Model, http://martinfowler.com/bliki/AnemicDomainModel.html, 2003

[10] 金田重郎，井田明男：存在従属に基づくユースケースの逆生成，電子情報通信学会，知能ソフトウェア研究会，5 月，2015 年

[11] マーチン・ファウラー：エンタープライズ アプリケーションアーキテクチャパターン,翔泳社，2005

[12] ロバート・C・マーチン：アジャイルソフトウェア開発の奥義 第 2 版 オブジェクト指向開発の神髄と匠の技，ソフトバンククリエイティブ，2008

[13] エリック・エヴァンス：エリック・エヴァンスのドメイン駆動設計，翔泳社，2011

[14] ヴァーン・ヴァーノン：実践ドメイン駆動設計，翔泳社，2015