ソフトウェアアーキテクチャの授業での取り上げ方について

小林　洋
東海大学
koba @ tokai.ac.jp

要旨

最近，ソフトウェア開発において重要性の認識が高まって来ているソフトウェアアーキテクチャを大学の授業で教えようとする，一つの取り組みについて述べる．まず，開発経験が無いといってもいい学生に，ソフトウェアアーキテクチャについて講義しようとする場合の問題点について述べる．次に，講義や実習の授業でソフトウェアアーキテクチャを取り上げようとした場合の問題点と課題について，授業での今までの試みを踏まえて述べる．なお，ソフトウェア開発の実習においては，ゲーミフィケーションの導入は，学生の興味をひくという点から効果的であった．

1. はじめに

ソフトウェア開発におけるアーキテクチャの重要性が認識されるようになって来ているため，情報系の学科では，何らかの形でそれを意識した授業を行っているものと思われる．しかしながら，通常，発表されるものは，どうしても，そのコンテキストも含め優れた事例ばかりになってしまい，一般的な状況が正しく伝わっていない恐れがあるように思われる．本稿では，一事例に過ぎないかもしれないが，比較的一般的と考えられる状況下で，ソフトウェアアーキテクチャを教えようとする取り組みを踏まえた上で，問題点や課題について述べる．問題点としては，まず，ソフトウェアアーキテクチャという用語には多義性があるため[1]，列挙して解説するだけでは，理系の学生の多くは興味を示さない上，混乱を招く恐れがあることが上げられる．学生には実際の開発経験など無いといってもいいため，単なる用語の羅列や，ボックス型の図形に単語が書かれただけのものと受け止められてしまうところもある．次に，大学と言う特性上，学問的にも扱いたいところだが，定量的評価方法が未だ確立されていないために，これが難しいことが上げられる．更に，単なる概念的な知識としてだけではなく，理解を深めてもらうためには実習が効果的であるため，最近，開発の実習において効果的な方法として注目を集めているプロジェクトベーストラーニング(Project-Based Learning :PBL)で行い，この中にソフトウェアアーキテクチャを取り入れることが考えられるが，PBL を実現させるためには，現状，予算や人員の確保等いくつかの課題がある．また，学部生の段階では，多くの学生は PBL よりも個人レベルでのソフトウェア作成力の向上の方が優先事項のように見受けられる．その場合，個人レベルの実習の問題となると，どうしても極めて小規模な問題にならざるを得ないが，その中でアーキテクチャをどのように取り入れるかは継続した課題である．なお，ソフトウェア開発の実習では，ゲームの要素を取り入れ，段階的にクリアして行けるようにすると，興味を持って取り組んでくれることが解った．この試みの成果としては，特に三層モデルでの開発により，ソフトウェア開発では動くものをただ作れば良いという訳ではなく．アーキテクチャを考慮する必要のある事を，認識してもらうことが出来たことではないかと考えている．

ここで取り上げた科目は，「情報システム開発・同演習」という3年生対象の週2コマ，概ね1コマずつ講義と実習を行う授業であり，単なる実習だけの授業ではない[2]．受講者は，前提条件としてオブジェクト指向と Java 及び SQL は履修済みであることとしている（但し，拘束力はない）．受講者数は，選択科目ということもあり，年度によって30名程度から80名程度と大きく変動する．授業は教員1名と大学院生の教育補助学生（Teaching Assistant: TA）2名で行っている．本稿は，主にこの授業での試みとその結果を基にした所見である．以下，2.では，ソフトウェアアーキテクチャの問題点について，3.では授業の取り組みとその問題点と課題について述べる．

[1] 観点の多様性と捉えても良いが，多くの学生から見れば実質同じ事かもしれないと思われる．

[2] 実習だけの授業では2コマで2単位，同演習とついている科目は，講義と実習からなり，2コマで4単位となる．TA の人数は，大学内の規定で定められている．なお，カリキュラム策定者と授業実施者は同じとは限らない．

2. ソフトウェアアーキテクチャの問題点

2.1. 用語の多義性

ソフトウェアアーキテクチャは，ソフトウェアの開発規模が大きくなり，ソフトウェア危機が提唱されるようになった 1960 年代後半頃から研究され始めたようであり，その定義は対象とする範囲も含め研究者や文献により様々である[3] [1]. これらの定義を列挙したものが，例えば SEI の Software Architecture の Web ページ[2]に記載されている. 狭義にはソフトウェアアーキテクチャは，プログラムの静的構造のみを指す場合もあるが，一般的には動的構造（振る舞い）も併せて指す. これらの表現方法としては図式が良く用いられ，前者は機能分割図や UML のクラス図等が，後者は UML のシーケンス図等が良く用いられる. これらに加え，表形式が補助的に使われ，厳密性を要求される場合には数学的表現が用いられる場合もある. これらの構造で，ソフトウェア開発者によって良く使われるものをパターン化したものに，細かい粒度では GOF のデザインパターン[3]等がある. また，より粗い粒度のミドルウェア的な構造としては MVC(Model-View-Controller) モデルや三層(Tree Tier [Layer])モデルなどがあり，ブッシュマンのソフトウェアアーキテクチャ[4]等に示されている. 他方，業務アプリケーションのパターンに対応したものとしては，ピーター・コードのビジネスオブジェクトモデリング[5]等に示されている.

ソフトウェアアーキテクチャについて定義した IEEE1471[6]（ANSI/IEEE 1471-2000：現在では ISO/IEC/IEEE42010:2011[7]）では，「ソフトウェアアーキテクチャとはコンポーネント，コンポーネント間およびコンポーネントと環境との関係，並びにその設計や発展性を導く原理によって具現化されるシステムの基本構成」と記されており，更に，アーキテクチャでは，利害関係者の関心である，性能，信頼性，セキュリティ，分配，発展性等を考慮する必要があるとしている. IEE1741 での定義の後半の部分については広義に解釈すると，開発の方法論，開発環境，技術標準，オントロジ（語彙体系）や原則，ガイドラインまで含むことになり，TOGAF（The Open Group Architecture Framework）[8-9]などではそのような観点で記述されている. また，情報システムの構築を策定しようとする場合，利害関係者（ステークホルダ）の立場により，情報システムを複数の観点（ビュー）から捉えることになる[10]. 対象とする業務（事業）の構造ということを名称にも明示したエンタープライズアーキテクチャ（EA）[11-13]では，図1に示すように，ビジネスアーキテクチャ，データアーキテクチャ，アプリケーションアーキテクチャ，及びテクノロジーアーキテクチャの4つで構造を捉えていて，対象業務の分析やシステム方式設計等を含んだアーキテクチャとなっている. 表記方法については，ビジネスアーキテクチャは，組織構造図と業務（ワーク）フロー（UML で表すなら，アクティビティ図やユースケース記述），データアーキテクチャは ER 図や DFD[4] ，アプリケーションアーキテクチャはモジュール構造図と振る舞い図（UML では，クラス図，コミュニケーション図，シーケンス図），テクノロジーアーキテクチャは構成図（UML では，配置図）などが使われている. なお，最近話題になった自動車向けの機能安全規格である ISO26262[14]では，ソフトウェアアーキテクチャについては，設計における手法として表記法，エラー検出と処理，および検証が示されており，安全性のための開発手法やシステムに必要な安全機構（機能）という観点から捉えている. このように，ソフトウェアアーキテクチャの定義は，狭義のものから広義のものまで様々なものがあり，コンテキストによって使い分けられている.

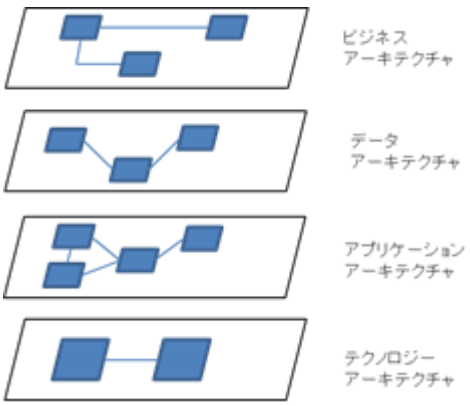

図1 エンタープライズアーキテクチャ

[3]用語の定義というものは時代と共にだんだん広義なものになって行く傾向があるが，それと共に曖昧性が増す. そのため，場合によってはやがて別の用語が用いられるようになる.

[4] DFD は，データアーキテクチャを表すとされる場合と，アプリケーションアーキテクチャを表すとされる場合があり，これは観点の違いと考えられる.

2.2. 評価方法

ソフトウェアアーキテクチャの評価の必要性については，連邦エンタープライズアーキテクチャ FEA（Federal Enterprise Architecture ）でも，FEA consolidated Reference Model Document Version2.3(2007)[12]で，Performance reference Model(PRM)に記載されている．但し，この文書に示されている評価尺度（Measurement indicator）については，カテゴリとグループで分類しているのみであり，具体的な評価尺度は各機関で作成する必要があるとされている．従来，ソフトウェアアーキテクチャの評価手法としては，いくつかのものが提案されており[15-22]，SAAM（Software Architecture Analysis Method）[21]や ATAM（Architecture Tradeoff Analysis Method）[22]が比較的良く知られている．SAAM では，アーキテクチャの修正容易性，拡張性等の開発の際の特性を，変更のシナリオを用意し，変更させるモジュール数，修正コスト等をスコアリングにより評価している．SAAM を拡張した ATAM では，アーキテクチャが性能や信頼性等の品質特性に関する要求を満たし得るかを，品質特性（性能，稼働率，修正容易性等）を列挙したユーティリティツリーに基づき，品質特性に関わるシナリオを用意し，要求を満たすかを評価している．いずれも，アーキテクチャを，シナリオに基づくシステムの振る舞いの妥当性によってスコアリングにより評価するという方法である．しかしながら，これらの手法は，人的・時間的コストの問題から，開発現場において用いる場合においては，簡略化した手法が用いられることが多いようである[23].

3. 授業の取り組みとその問題点と課題

3.1. 授業の主な内容

授業の主な内容は，以下の通りで，（1）から（6）は，（7）を行うための準備教育である．

（1）ソフトウェアアーキテクチャと UML による設計
（2）部品化を考慮したプログラミング
（3）GUI プログラミング
（4）Java と SQL の連接
（5）GUI から DB までの連接
（6）Web プログラミング
（7）ロバストネス分析手法による総合演習

3.2. 講義での問題点と課題

ソフトウェアアーキテクチャを広義にとらえると，情報システム開発に関連する事柄の，あらゆる構造的なものとなってしまう．このような内容は，極めて小規模の練習問題程度のプログラムしか開発経験の無い学生にとっては，実感の伴わない概念だけの話になってしまう．ソフトウェア開発の分野でしばしば用いられる前述の図1のような箱型のモデル記述は，一時限りの暗記事項で終わってしまう可能性がある．また，ソフトウェア開発に関わるアーキテクチャの定義はいろいろあると言って，列挙して示したりすると学生の多くは混乱するだけのようである．初学者にとっては，まず，講義では定義のごく数種類のみを示し，多義性については，最後に軽く触れる程度に留めるのが良いようである．しかしながら，例えば MVC モデルに限定した話でも，図２に示すように，Smalltalk 流の MVC と Web アプリケーションでの MVC2 では振る舞いも異なるし，C や M の範囲ついては，対象や研究者によって解釈の違いが見られる．MVC2 の捉え方では，C は単純な振り分け処理のみで，M はアプリケーションのみ，或いは永続データへのアクセスの連接も加える場合があり，更には永続データも含める場合がある．MVC モデルと共に図3に示す三層モデル（プレゼンテーション層：P，ビジネスロジック層：B，データ層：D）を教えようとする場合，この二つについては，単に観点の違いであると説明する方法があるが，それで納得してもらえるかは疑問である．そこで，図4のように，V+C＝P，M=B+D（又は M=B）であると説明する方法も考えられるが，このように限定したケースであっても，学生に理解してもらうには，いかにしたら良いかは，課題であろう．

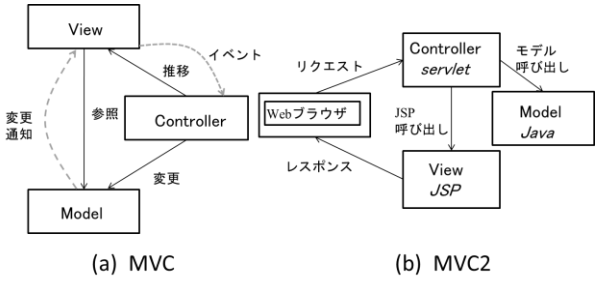

図2 MVC モデル

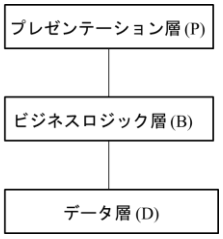

図3 三層モデル

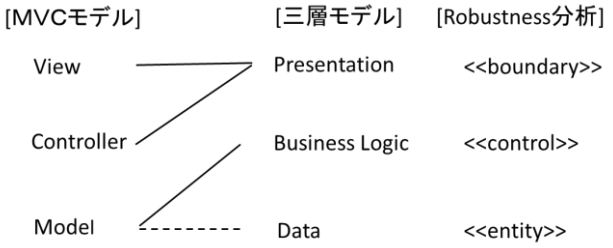

図4 MVC と三層（PBD）モデルとの対応？

3.3. 実習での問題点と課題

(1) 開発工程に沿った実習

a. PBL と個人のスキル

ソフトウェアアーキテクチャを学生により深く理解してもらうには，具体的なソフトウェア開発の問題により，開発工程に沿って要求分析からプログラミング[5]までの実習の中で理解してもらうのは効果的であると考えられる．このような実習の方法として，最近，数人（例えば5～6人程度）の学生でグループを組んである程度の規模のソフトウェアを開発するというプロジェクトベーストラーニング（Project-Based Learning: PBL）[26-28]が注目を集めている．PBL の実施例を見ると，グループのリーダの力が重要となるということで，企業等から招へいした実開発経験者を各グループに一人ずつ配置し，プロジェクトの全般管理の他，業務の適切な割り当てや，ソフトウェアの不具合についての適切なサポートを行ってもらい，更に，各グループに大学院生の TA も配置しているケースが多く見受けられ，それが理想なのであろうと思われる．しかしながら，現実には，大学によっては，まず予算や大学院生の数の問題により，そのような十分なサポート体制が取れない場合がある．また，学部生の多くは3年生であっても，未だソフトウェア開発についての個人の知識・技能の向上を行う段階にあり，プロジェクトを行える段階ではないように見受けられる．現状では，大学により状況は異なるかもしれないが，学部生の内は，各人のスキルの向上を図るために，個人ごとに設計からプログラム作成まで行う方が適しているように思われる． この場合，現在の大学では，基本的には「ほとんどの者がほぼ達成できるような問題」を出題することが求められているようなので，授業時間という制約もあり，トイレベルの極めて小規模な問題にせざるを得ない．但し，学生数だけ異なる問題を用意するのは，現実的に不可能であるので，共通又は数種類の問題で実習を行わざるを得ない．いずれにせよ，従来型の個人ベースの実習においてもであるが，特に，PBL の実施にあたっては，予算と人員の確保が第一の課題である．

b. 開発対象とする問題の選定

学生を対象としたソフトウェア開発の実習においては，上流工程の要求分析やシステム設計においては，対象業務についての知識が必要となる．すると，教育の時間的制約からどうしても履修登録システムや図書館システムなど，学生になじみの深いもので行わざるを得なくなる．但し，ここで問題になるのは，このようなシステムに限らず，今の時代，日常的な業務系システムにおいてはほとんど既存のものが存在するため，システム設計においては，学生は既存のシステムの処理手順やそれに伴う画面の流れの影響を受けてしまい，単にそれらの一部の再現になってしまう可能性が大きいという問題がある．履修登録システムや図書館システムも含め適切な対象業務を選び，適切な規模の問題を作成することは，課題である．

(2) アーキテクチャを意識した開発の実習方法

学生に，小規模の問題にせよ，上流工程からプログラミングまで通して修得してもらい，更にその中で三層モデルの一種についても理解してもらうために，次のような手順の実習を実施している．但し，受講者は，前提としてオブジェクト指向と Java 及び SQL は履修済みであることとしているが拘束力はないので，最初の数回の授業で，確認のための実習を行っている．また，実習の効率化のために，GUI や JDBC 等のサンプルプログラムをファイルの形で与え，これを活用してもらっている．なお，コーディング規約については，学生がうんざりしない程度の簡単なことのみに留めている．

まず，図5に示すように，要求仕様に相当する文書（あまり長くないもの）を与え，段階的に成果物を作成して行けるように，ロバストネス分析手法[24-25]を簡略化した次のような手順で実習を試みている[6]．最初は，①UML のユースケース（図と記述）を書いてもらう[7]．（時間的余裕があれば，更に業務フローを表すためのアクティビティ図

[5]授業時間という制約を考えると，テスト工程まで十分に行うのは困難なためテスト工程は簡略化せざるを得ない．また，学生の多くが興味を抱くのは，作成の部分である．

[6] ロバストネス分析手法を採用した理由は，画面，ロジック，データの分離を教えるのに効果的と思われたためである．

[7]実務においては，ユースケースをいきなり書くようなことは，あまり有り得ない事かもしれないが．

及び粗いシーケンス図を書いてもらう.)次に,②画面のデザインのスケッチと画面フロー図(画面遷移図)を書いてもらう.更に,③文献[5]等に示されている概念モデル(ドメインモデル)を参考に必要なクラスを考えてもらい,ロバストネス分析手法での boundary(View), control(ロバストネス分析では,アプリケーションロジックを指す), entity(Model:ロバストネス分析では,永続的データを指す)に分けてクラス図を作成してもらう.その結果を,④画面デザインと画面フロー図にフィードバックし,必要な修正を行ってもらう.

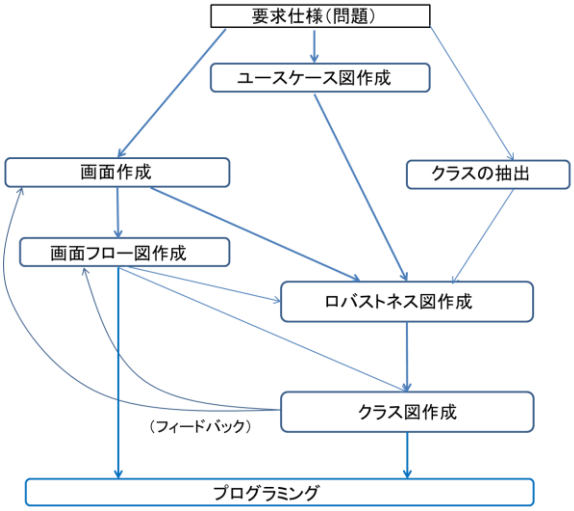

図5 ロバストネス分析の簡略手法

すると,画面デザインや画面フロー図については,インターネットやスマートフォンが普及した現在,Web 画面や GUI 画面についてのイメージが湧きやすいのか,比較的容易に考えられるようである.

クラス図の作成については,参考となるクラス図のカタログがビジネスオブジェクトパターンとして文献[5]等に示されているので,これらを参考にして考えてもらうのが実際的なように思われる.初学者には,いきなり,「販売」や「注文」がクラスになるということなどは,まず思いつかない.

最後に,⑤実装を行ってもらう.実装は,boundary, control, entity に分けて記述したクラス図と画面デザイン及び画面フロー図を参考に Java と SQL(本授業では SQL Server)で行っている.ところで,永続データ部分の実装については,リレーショナルデータベースの使用が一般的であろうが,この際,クラス図からテーブルへの変換が必要となる.この場合,クラス図が基本的な記述のみの単純なものであれば,正規化がうまく行われるかは別として,テーブルへの変換は比較的容易に行える.

もちろん,このような小規模の仕様の場合には,実際の開発では,三層への分離など不要かもしれないが,学習としては行っても良いと思われる.

以上については,現在,学生の学習状況を見ながら試行錯誤して行っていることである.学生の実習の場合,学生の習熟度や授業の時間,更には予算の問題もあり,企業で行われているような実際的な手法の一つを,そのまま導入すれば良いという訳ではなく,学生の状況に合わせた修正が必要と思われる[8].

(3) 設計書とプログラムのアーキテクチャのギャップ

プログラムの開発経験の乏しい学生には,設計図を書くだけでは,その設計図が実際に動くプログラムに繋がるという実感が湧きにくく,プログラム作成までを経験するのが望ましいと思われる.ところで,学生が上流工程からプログラミングまでを通して行おうとすると,設計図は何とか出来ても,プログラミングの段階になると,設計図のとおりではプログラミング困難ということで,設計図を棚上げして,プログラミング容易なものを作成してしまうことが起こり得る.そして,完成したプログラムと設計図と突き合わせてみると,図6のように,ソフトウェアアーキテクチャ的にもかなり違ったものが出来ているということが起こる.しかし,この事もまた貴重な体験なのであろうから,とにかく,一度,上流工程からプログラミングまで通して開発してみること自体が重要なのだとも思われる.但し,学生の多くは,問題には唯一の解答があり,それに向けて一直線に開発を進めたいと考える傾向があり,試行錯誤を嫌がる傾向にある.

図6 仕様とプログラムの不整合

[8] 企業から大学に移った教員が,当初,「このような授業の内容は現場ではやっていない.」と言うのを時々耳にするが.

また，最近では，画面設計自体は，GUI ツールの発達により，見栄えの良いものが手軽に出来る．しかし，そのまま，プログラムの設計や作成を行ってもらおうとすると，作成できなかったり，又は，できたとしても，画面のプログラムにロジックやデータベースとのアクセスまで張り付いたものになり易い．GUI の作成だけが先行してしまうのを，いかにして解消するかは課題と言えよう．

(4) デザインパターンの初学者にとっての解り難さ

最近では書籍の付録やネットに，プログラムのサンプルが多数掲載されている．プログラムの問題を考える前に，サンプルを捜すというアプローチをとる学生も少なくない．但し，これらのプログラムは，お手本として載せているせいか，ある程度の熟練者が作ったものが多く，巧みに部品化され，デザインパターンが用いられているものも多い．ところが，初学者にとっては，これがプログラムの理解の妨げになってしまうことがある．例えば GOF の Abstract Factory パターンなど，初学者にとっては，面倒なだけのように思えるし，Composite パターンなどもプログラムを見ると，戸惑うようである．一般的に，熟練者の作った筋の良いプログラムは，初学者にとって必ずしも解り易いプログラムではない．もちろん，熟練者が巧みに用いているデザインパターンを学習することによって，プログラミングの技能向上を図るというメリットはあるだろうが，多くの学生にとっては，プログラミングが難しく思えてしまい，苦手意識を持つことになるというデメリットも大きいように思える．デザインパターンをどのように扱って行くかは課題であると考えられる．

なお，初学者にとっては，パターンを用いた場合のメリットも解り難いようであるが，一つの解決策として，結果の可視化が考えられる．例えば Singleton パターンの効果については， 図7のように，画面遷移において，ボタンを複数回クリックすると画面が複数個作成されてしまうのを Singleton パターンで防止することが出来ることなどは，学生にとっては直観的で解り易いようである．

図7 シングルトンパターンの振る舞いの可視化

(5) アーキテクチャの評価方法とその実習方法

大学では，学生に対象を工学的に評価する手法，できれば定量的評価する手法を取り上げて教えたいところである．ソフトウェア開発における，広い意味でのアーキテクチャの評価手法を上げるならば，現在は ATAM が良く知られており，これから派生したような研究も多い[15-22]．しかし，これらの手法については，そもそも，企業等が実開発で用いる場合にも課題があるようである．まず，アーキテクチャの評価は開発前に行いたいが，詳細な設計もされていない段階で，妥当な評価をいかにして行うかということが上げられている．次に，ATAM 等を実施するには，マンパワー（コスト）がかかるので，それをいかに低く抑えるかという点であり，実開発においては，より簡略化した手法を用いているのが実情のようである[23]．このような状況にある評価手法を授業で取り上げるのが適切かどうかも疑問のあるところだが，授業で取り上げた場合，学生に理解してもらうには，適当な規模の具体的問題の実習により行う必要があると考えられ，問題の作成と併せて，良い解答例の作成も課題となる．

(6) ゲーミフィケーション

実習教育においては，ゲームになじんでいる学生が多いことから，ゲームのステージ達成という要素を取り入れた，インクリメンタルな開発という方法が効果的なようである（図8参照）．

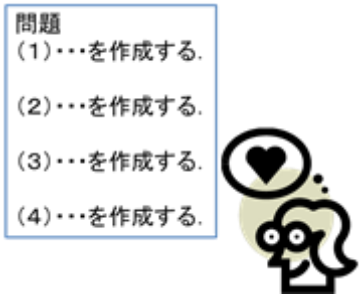

図8 ゲーミフィケーション（小問（ステージ）への分割）

本稿で対象とした授業の場合には，3.1.で示した授業の内容の実習において，各問題では可能な限り小問に分割することを試みた．また，各問題自体が総合演習で，そのままではないにせよ，活用できるような問題となるようにした．実習においては，問題を分割し，段階的に作って動くのを確認しながら，徐々に拡張して行くように問題を作っておくのは効果的であった．ゲーム慣れしている学生にとっては，ソフトウェア開発において，ゲームのス

テージのようなものを設けて，適宜与えられたヒントを参考にしながら，1段階ずつクリアして行くという方法，つまりプログラム開発のゲーム化（ゲーミフィケーション）[29]という方法が，興味を持ってもらえ有効であった．但し，このように小問に分割する方法では，どうしても各小問については，解釈が限定されるような問題になってしまう．本来は，上流工程で仕様が曖昧な部分は各人の判断に任せ，出来るだけ自由に作成してもらいたいところだが，小問の連続にすると自由度が失われ創造性が妨げられてしまう恐れがあるという問題がある．しかし，プログラム開発での不具合に教員やTAが対処可能で，ほとんどの学生がほぼ達成できるようにするためには，やむを得ないと考えている．開発の実習のゲーミフィケーションについては，他にもいろいろな手法の導入が考えられるだろうが，これは今後の課題と考えている．

4. おわりに

本稿では，ソフトウェアアーキテクチャを，大学で教えようとする場合の問題点と課題を，現在の授業の状況を踏まえて述べた．

問題点としては，そもそも，ソフトウェアアーキテクチャという用語があまりにも多様に使われていることと，定量的評価法が確立していないことが上げられる．

授業を講義として行う上での課題としては，実開発の経験の無い学生に，対象や研究者によって解釈の異なる場合のある抽象化・モデル化した内容をどのように理解してもらうかということが上げられる．

実習として行う上での課題としては，まず，実習の一般的な課題として，予算と実習補助者をいかに確保するかということが上げられ，特に，PBL においては，切実な問題となる．なお，PBL については，現状，一般的な状況においては，学部生の段階では，個人の知識・技能の向上を行う段階にあるように思われるため，卒業研究や大学院あたりでの実施の方が適切なように思われる．

実習を，アーキテクチャを考慮したものにしようとする場合，実習では，問題を極めて小さなものにせざるを得ないため，三層モデルの場合，三層モデルを適用した小さな適当な問題の作成ということが，まず課題として上げられる．また，作成するシステムは，時間的制約やサポート体制のことも考えると，現状，学生にとって仕様が理解容易な，限られた種類のものになってしまう傾向があり，その意味でも適切な問題の作成というのは課題である．

開発においては，最近では，GUI の作成が容易になったために，GUI の作成だけが先行してしまう傾向があり，これをいかに解消するかも課題であると考えられる．また，学生が書籍やネットのプログラムのサンプルを参考にするのは止むを得ないと思われるが，熟練者が作成した巧みに部品化したデザインパターンを使ったプログラムは，学生には理解しにくい事が多い．これにより，プログラミングに対して苦手意識を持ってしまう学生も多いようなので，デザインパターンを実習でどのように取り扱っていくかは課題と考えている．

更に，大学の授業においては，教えたことは理解度をテストし評価を行う必要があるため，実習において，どのようなテスト問題で理解度を評価するのが妥当であるかということも，大きな課題である．

以上のように，ソフトウェアアーキテクチャを取り上げようとした授業の試みは，今の所，課題が多く残されているが，成果として上げられる最大のことは，ソフトウェアの開発においては，ただ動くものを作成すれば良いというだけではなく，アーキテクチャを考慮する必要があるということを，学生に認識してもらい，多少なりとも納得してもらう事ができたことであろうと考えている．なお，実習で用いたロバストネス分析手法は，三層の分離を強調するという意味では，有用であると思われる．

また，実習においては，問題を小問に分け，段階的にクリアして行くようにするという，一種のゲーミフィケーションの手法の導入は，実習の促進に効果的であった．ゲーミフィケーションについては，他の手法の導入の検討が今後の課題として残されている．

授業の工夫とその効果については，授業アンケートや実習問題の提出状況等を基に定量的評価の形で示した方が良いのであろうが，公表できるような形にはなっていないので，本稿では，授業の状況を基にした所見のみに留めた．社会的な影響も考慮した上で，外部へ公開できる形での客観的データの収集と分析評価については，今後の課題と考えている． 但し，最終的には，学生がソフトウェア開発に悪いイメージを持たず，ソフトウェア開発業界を避けないようにすることが，大きな課題であると考えている．

参考文献

［1］ 岸知二，野田夏子，深澤良彰：ソフトウェアアーキテクチャ，共立出版，2005.
［2］ SEI Software Architecture:

http://www.sei.cmu.edu/architecture/start/glossary/community.cfm
[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns, Addison-Wesley,1995. [邦訳]本位田真一, 吉田和樹監訳, オブジェクト指向における再利用のためのデザインパターン,ソフトバンク, 1995.
[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal: Pattern-Oriented Software Architecure: A System of Patterns, John Wiley & Sons, 1996. [邦訳]金澤典子他訳,ソフトウェアアーキテクチャ:ソフトウェア開発のためのパターン体系, 近代科学社, 2000.
[5] P. Coad, D. North, and M. Mayfield: Object Models: Strategies, Patterns, and Applications 2nd Edition, Yourdon Press, 1997. [邦訳]依田光江訳, 戦略とパターンによるビジネスオブジェクトモデリング, ピアソン, 1999.
[6] ANSI/IEEE1471-2000:http://standards.ieee.org/findstds/standard/1471-2000.html/.
[7] ISO/IEC/IEEE42010:2011: http://www.iso-architecture.org/42010/.
[8] TOGAF: http://www.opengroup.org/togaf/.
[9] オープングループジャパン: http://www.opengroup.or.jp/togaf.html/.
[10] N.Rozanski, and E. Woods: Software Systems Architecture: 2nd Edition,2012.[邦訳]榊原彰監訳, 牧野祐子訳, ソフトウェアシステムアーキテクチャ構築の原理:第2版, SB クリエイティブ, 2014.
[11] FEA: A Practical Guide to Federal Enterprise Architecture Version1.0 :http://www.enterprise-architecture.info/Images/Documents/Federal Enterprise Architecture Guide v1a.pdf,2001.
[12] FEA consolidated Reference Model Document Version2.3:http://www.whitehouse.gov/sites/default/files/omb/assets/fea_docs/FEA_CRM_v23_Final_Oct_2007_Revised.pdf ,2007.
[13] 萩原正義:アーキテクトの審美眼, 翔泳社, 2009.
[14] ISO26262: http://www.iso.org/iso/catalogue_detail?csnumber=43464.
[15] U. v. Heesch, V-P. Eloranta, P. Avgeriou, K. Koskimies, and N. Harrison: Decision-centric Architecture Reviews, IEEE Software,Vol.30, No.1, pp.69-76, 2014.
[16] L. Bass, and R. L. Nord: Understanding the Context of Architecture Evaluation Methods, Proc. 2012 Joint Working Conference on Software Architecture & 6th Europian Conference on Software Architecture, pp.277-281, 2012.
[17] L. Dobrica, and E. Niemela: A Survey on Software Architecture Analysis Methods, IEEE Trans. Software Engineering, Vol. 28, No.7, pp.638-653,2002.
[18] J. Knodel, M. Lindvall, D. Muthig, and M. Naab: Static Evaluation of Software Architectures, Proc. Conference on Software Maintanance and Reengineering(CSMR'06),pp.284-294, 2006.
[19] N. Harrison, and P. Avgeriou: Pattern-based Architecture Reviews, IEEE Software, Vol.27, No. 6, pp.66-71, 2011.
[20] P. bengtsson, N. Lassing, J. Bosch, and H. v. Vliet: Architecture-level modifiability analysis (ALMA), The Journal of Systems and Software, No. 69,pp.129-147, 2004.
[21] R. Kazman, L. Bass, M. Webb, and G. Abowd; SAAM: A Method for Analyzing the Properties of Software Architectures, Proc. 16th ICSE, pp.81-90,1994.
[22] R. Kazman, M. Barbacci, M. Klein, and S.J. Carriere: Experience with Performing Architecture Tradeoff Analysis, Proc. 21st ICSE, pp.54-63, 1999.
[23] T. エンゲルバーク著, 長谷川裕一, 土岐孝平訳: 間違いだらけのソフトウェア・アーキテクチャ:非機能要件の開発と評価, 技術評論社, 2010.
[24] D.Rosenberg, and K.Scott: Use Case Driven Object Modeling with UML: A Practical Approach, Addison-Wesley, 1999.[邦訳] 長澤嘉秀, 今野睦訳:ユースケース入門ーユーザマニュアルからプログラムを作る, ピアソン, 2001.
[25] C. T. Arrington: Enterprise Java with UML, Wiley, 2002.[邦訳]平川章他訳:UML によるエンタープライズ Java 開発, 翔泳社, 2002.
[26] B.J.S. Barron, D. L. Schwartz, N.J. Vye, A. Moore, A. Petrosino, L. Zech, and J.D. Bransford: Doing with understanding: Lessons from research on problem- and project-based learning, The Journal of the Learning Sciences, Vol.7, pp.271-311, 1998.
[27] 松澤芳昭, 大岩元:産学共同の Project-based Learning によるソフトウェア技術者教育の試みと成果, 情報処理学会論文誌, Vol.48, No.8,pp.2767-2780, 2006.
[28] 鶴保征城, 駒谷昇一:ずっと受けたかったソフトウェアエンジニアリングの授業1, 2[増補改訂版], 翔泳社, 2011.
[29] 井上明人:ゲーミフィケーション―"ゲーム"がビジネスを変える, NHK 出版, 2012.