③ 酒屋問題再考
―新たな共通問題の作成を目指して―

応

丸山勝久（立命館大学情報理工学部）

酒屋問題の功績と再考の必要性

「酒屋問題」を知っていますか．正確にいうと，これは，1984年の4月に開催されたシンポジウム「プログラム設計技法の実用化と発展」で提示された共通問題[1]のことを指す（**図-1**）．いわゆる「お酒の在庫管理システム」の設計問題である．

1984年の本誌9月号と11月号において，この問題を扱った合計9編のプログラム設計技法の適用事例が掲載されている[1),2)]．さらに，この共通問題は一部改訂され，プログラミング・パラダイムの相違点の明確化という観点から，1985年5月の特集において6つのプログラム設計・記述技法の適用が試みられた[3]．当時はプログラム設計技法が次々に登場した時期であり，これらの特集に掲載された適用事例を通して設計技法を知った，あるいは学んだという研究者や実務者も数多く存在したと予想される．また，上記の特集以外でも，この問題を目にした読者は多いだろう．

このように，酒屋問題はプログラム設計技法の比較という観点において共通の課題を提供することで，さまざまな設計技法に関する情報共有や議論を促進するという重要な役割を果たしてきた．しかしながら，ソフトウェア工学コミュニティからは，必ずしもこの酒屋問題が現在の共通問題として十分とはいえないのではないかという疑問があがっている．

たとえば，ソフトウェア開発におけるプログラム設計の位置付けは大きく変化しており，これだけを分離して取り扱う共通問題では，ソフトウェア工学研究のごく一部しか比較や検討の対象とはできない．また，当時に比べてソフトウェア工学は大きく進展し，再利用や進化を強く意識した設計手法や管理手法が登場している．さらには，ソフトウェア工学の適用領域の拡大は著しく，生産性と品質の向上だけがソフトウェア工学の目指すべき目標という考え方は時代遅れとなっている．つまり，多様な利害関係者を巻き込むソフトウェア開発と特定の機能を満たすプログラム作成との違いが，当時より顕著になってきている．このため，プログラム設計に焦点を当てた共通問題では対処できない場面が数多く発生している．

ある酒類販売会社の倉庫では，毎日数個のコンテナが搬入されてくる．その内容はビン詰めの酒で，1つのコンテナには10銘柄まで混載できる．扱い銘柄は約200種類ある．倉庫係は，コンテナを受け取りそのまま倉庫に保管し，積荷票を受付係へ手渡す．また受付係からの出庫指示によって内蔵品を出庫することになっている．内蔵品は別のコンテナに詰め替えたり，別の場所に保管することはない．

空になったコンテナはすぐに搬出される．

積荷票：コンテナ番号（5桁）
　　搬入年月，日時
　　内蔵品名，数量（の繰り返し）

さて受付係は毎日数10件の出庫依頼を受け，その都度倉庫係へ出庫指示書を出すことになっている．出庫依頼は出庫依頼票または電話によるものとし，1件の依頼では，1銘柄のみに限られている．在庫が無いか数量が不足の場合には，その旨依頼者に電話連絡し，同時に在庫不足リストに記入する．また空になる予定のコンテナを倉庫係に知らせることになっている．倉庫内のコンテナ数はできる限り最小にしたいと考えているからである．

出庫依頼：品名，数量
　　送り先名

受付係の仕事（在庫なし連絡，出庫指示書作成および在庫不足リスト作成）のための計算機プログラムを作成せよ．

出庫指示書：注文番号
　　送り先名
　　コンテナ番号
　　品名，数量
　　空コンテナ搬出マーク
　　（の繰り返し）
在庫不足リスト：送り先名
　　品名，数量

- なお移送や倉庫保管中に酒類の損失は生じない．
- この課題は現実的でない部分もあるので，入力データのエラー処理などは簡略に扱ってよい．
- 以上あいまいな点は，適当に解釈して下さい．

図-1　酒屋問題（文献1）より引用）

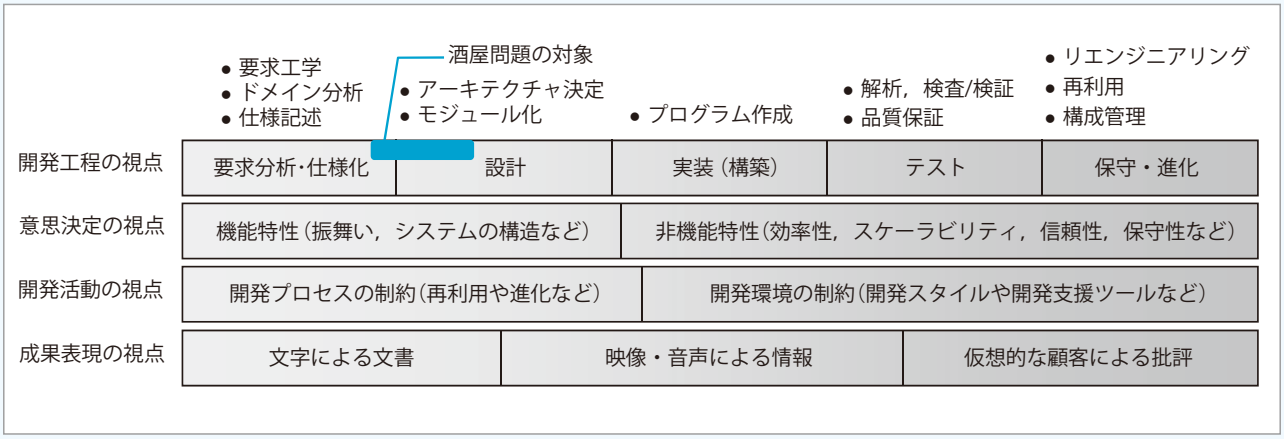

図 -2　ソフトウェア工学における共通問題を作成する際の視点

共通問題を作成する前に

ここでは，現在のソフトウェア工学研究の評価という視点で当時の酒屋問題を眺めて，新たな共通問題を作成する際に考慮すべき 4 つの視点（**図 -2**）を議論する．

❖ 開発工程の視点

酒屋問題を見てみると，要求（ソフトウェアを利用することで実現したい内容）と仕様（要求を実現するためにソフトウェアが満たすべき制約条件）が混在していることに気付く．つまり，図 -2 に示す要求分析・仕様化工程における成果物を用意し，そのすぐ後の工程において設計手法を考えさせるようになっている．仕様の変更（解釈の自由）を許しているため，要求の仕様化を対象とした問題と捉えることもできるが，設計を主な対象としていると考えて差し支えないだろう．

それでは，ソフトウェア工学研究の成果を評価するために新たに作成する共通問題は，設計技法だけを対象とするので十分であろうか．この問いに答えるのは簡単で，「ノー（いいえ）」である．

確かに，設計は問題（problem）領域（要求，解決すべき課題，システムの外部環境）と解（solution）領域（実装，課題を解決する手段，システムの内部環境）をつなぐソフトウェア開発の中心的活動である．しかし，設計だけでソフトウェアが開発できるわけではない．むしろ，近年のソフトウェア工学では，図 -2 に示す設計（狭義の設計）だけを切り離して，それが問題領域と解領域をつなぐ唯一の手段と捉えていない．

たとえば，テスト駆動開発では，テストケース作成とテスト実行の繰り返しにより問題領域と解領域をつないでいる．また，リファクタリングのように従来は保守技術に分類されていた活動を上流工程で積極的に活用することで，開発者が問題領域と解領域を自由に行き来することが奨励されている．さらには，アジャイル開発のように顧客を開発に取り込む姿勢は，要求や要求仕様が設計や実装の前に与えられるというソフトウェア工学における古い常識を覆している．

ここまでをまとめると，当時の酒屋問題は狭義の設計に焦点を当てており，ソフトウェア開発を問題領域と解領域をつなぐ幅広い活動として捉えていない点で不十分である．ソフトウェア工学の共通問題を新たに作成する際には，要求を満たす成果物をどのように作り出すかという点で，開発の全工程が対象となることを意識することが重要である．その上で，その共通問題がどの開発工程における技法の比較や検討を目指しているのかを明確に示すことが求められる．この点に関しては，文献 4）の「図書館問題 2.0」が参考になる．この問題では，研究チャレンジという観点で，共通問題が対象とする分野が明確に示されている．

❖ 意思決定の視点

酒屋問題は大雑把にいうと，搬入されたお酒のコンテナを入力として在庫状況を遷移させる状態機械（状態遷移システム）と，入力となる出庫依頼と在庫状況に応じて出庫指示書を出力する状態機械の組合せで実現できる．イベントやアクションの詳細，在庫データの永続性を無視すると，近年のビジネスアプリケーションの基本形と見なせる．このことより，酒屋問題自体はソフトウェア工学の共通問題として汎用性を備えているといえる．酒屋というドメインになじみがなければ，インターネット上の会員制サービスや論文査読システムなどに比較的簡単に作り変えることができるだろう．

そうであれば，酒屋問題が現在の設計技法の比較や検討という観点で，このままでも十分利用可能であるかというと，残念ながら，そうとはいえない．一昔前のソフトウェア工学では，設計は仕様を実装に変換するための活動と捉えられていた．その上で，ソフトウェアに求められる機能を実現するための分割とモジュール化が，設計の主な関心事であった．これに対して，現在のソフトウェア工学では，設計は単なる変換活動ではなく，顧客の要求を実現する手段を見つけ出す意思決定活動であるという見方が自然である．これは，ソフトウェアの品質に影響を与える機能要求や非機能要求が，必ずしも設計作業の開始前にすべて確定している必要がないことを指している．設計を繰り返すことで機能要求や非機能要求が確定し，本当に必要なソフトウェアの姿や利用方法が見えてくるという発想である．

設計活動に対する見方が変わっていることを踏まえて，意思決定（設計判断）に関する制約や基準を導入することで，作成できるソフトウェア工学の共通問題の可能性が広がる．たとえば，セキュリティ機能の実現を考えた場合，分散的な利用を想定しているのか集中的な利用を想定しているのかで，ソフトウェア構造や実装技術は大きく異なる．このことは，セキュリティ機能の実現を共通問題で扱う場合，利用環境に関する制約が存在することを表している．利用環境を指定してソフトウェアアーキテクチャや実装における信頼性を比較する共通問題を作成することもできるし，利用環境をあえて指定せずセキュリティ機能の実現とスケーラビリティとのトレードオフを考えさせる（要求の見直しも含む）共通問題を作成することもできる．セキュリティ以外にも，並列性や例外処理などに関する制約を付加したり外したりすることで，さまざまな形態のシステム開発を題材とすることが可能である．

ただし，意思決定に関する制約や基準が曖昧なままでは，共通問題に対する成果物（問題の解）の評価が困難となる．共通問題を作成する際には，制約や基準に関する十分な検討が必要であり，その提示に工夫が求められるだろう．

❖ 開発活動の視点

これまでの 2 つは，開発するソフトウェアに対して制約や基準を適切に与えることで，共通問題を作る視点である．これに対して，3 つ目は，開発活動に対して制約を適切に設定することで共通問題を作る視点を示す．具体的には，再利用や進化などの開発プロセスに関する制約を指す．たとえば，将来の改変を予測した設計，類似ソフトウェアの構築を最初から意識した設計，過去の設計成果物が残っているというもとでの設計などが開発活動に対する制約として考えられる．

これらの概念は特に新しいものではない．しかしながら，現在のソフトウェア工学においては，ビルディングブロック（成果物の一部）を単純に再利用するという考え方から，経験やノウハウを再利用するパターンやドメイン知識全体（特定のドメインに関係する過去の開発知識）を再利用するドメイン特化開発に関心が移っている．さらには，ソフトウェア変更は原則避けるべきであるという考え方に基づき，既存ソフトウェアの変更を保守活動と捉えるのも時代遅れである．ソフトウェアは本質的に変化し続けるという前提に基づき，変更を積極的に取り込んだ開発技法への需要もますます大きくなっている．開発するソフトウェアに対する機能特性および非機能特性を同一にしたとしても，再利用や進化という

視点から開発技法を比較および検討できる共通問題が作成できる．

開発活動への制約としては，ほかにも開発スタイルや開発支援ツールという開発環境に関する視点が考えられる．残念ながら，過去のソフトウェア工学においては，開発における人間の協調作業にあまり関心が向けられていなかった．これに対して近年では，グローバルソフトウェア開発，オープンソース開発，ソーシャルソフトウェア工学という新しい分野が研究の対象として認識されている．また，近年の開発支援ツールは，今まで以上にソフトウェアの生産性や品質に大きく影響を与えるようになってきた．これらの点を考慮すると，協調支援や知識共有支援の有無，自動化ツールの有無や自動化の程度なども共通問題で取り扱う対象となり得る．

❖ 成果表現の視点

ソフトウェア開発が高度な知的活動であるという前提に立つと，共通問題における最終成果物を文字や図により書かれた文書だけで表現することには限界がある．たとえば，意思決定においてどのような思考や議論が行われたかを知るためには，開発の様子を録画し，その結果を比較および検討する方がよいかもしれない．実際に，このような試みがワークショップ用の題材提供のために実施され，その分析結果が報告されている[5]．また，共通問題への取り組みやその結果をソフトウェア工学教育に導入しようとする場合，文書よりも映像の方が有利であろう．共通問題の最終成果物はプレゼンテーションで行い，仮想的な顧客による批評まで含めるという方法も考えられる．

さあ共通問題を作成してみよう

共通問題を作成するために考慮すべき視点を整理した．ここに示した視点は筆者の個人的な考えに基づいており，必ずしも考慮しなければならないわけではないし，抜けがないともいえない．しかしながら，ソフトウェア工学の進展や拡大により，共通問題を作成する際に考慮すべき視点が増えていることは実感できたであろう．

共通問題を作成するためにさまざまな視点を考慮することは，ソフトウェア工学を客観的に見直すことにつながり，その研究や実践にとって決して無駄にならない．後生に残るソフトウェア工学の共通問題の作成にぜひ取り組んでほしい．

参考文献

1）山崎利治：共通問題によるプログラム設計技法解説，情報処理，Vol.25, No.9, p.934（Sep. 1984）．
2）山崎利治：共通問題によるプログラム設計技法解説（その 2），情報処理，Vol.25, No.11, p.1219（Nov. 1984）．
3）二村良彦，雨宮真人，山崎利治，淵　一博：新しいプログラミング・パラダイムによる共通問題の設計，情報処理，Vol.26, No.5, pp.458-459（May 1985）．
4）鵜林尚靖，亀井靖高：図書館問題 2.0：ソフトウェア工学研究における共通問題例，情報処理学会ウィンターワークショップ 2012 予稿集，pp.129-130（Jan 2012）．
5）Baker, A., Hoek, van der A., Ossher, H. and Petre, M. : Studying Professional Software Design, IEEE Software, Vol.29, No.1, pp.28-33（2010）．

（2013 年 5 月 29 日受付）

丸山勝久（正会員）maru@cs.ritsumei.ac.jp
1993 年早稲田大学理工学研究科修士課程修了．同年，日本電信電話（株）（NTT）入社．2000 年より立命館大学．ソフトウェア保守と進化，ソフトウェア開発環境の研究に従事．