∽研究∽

ソフトウェア開発における品質管理と目的物の完成について

中　地　　充*
中　原　國　尋**

Ⅰ　はじめに

現代社会におけるソフトウェアの役割は，その重要性が日増しに大きくなっている。ビジネスにおける様々な機器がコンピューターによって制御されているが，それを担っているのはソフトウェアである。ソフトウェアは，もともとは会計に関連する業務を中心にその利用が広がっていった[1)]が，現在では会計以外の業務にも広く利用されている。例えば，会社における交通費の精算システムや教育研修履歴の管理，情報共有のためのグループウェアなど，日々の業務に欠かせない仕組みがソフトウェアによって実現

＊弁護士，中央大学法科大学院実務講師，同法科大学院 2007 年 3 月修了
＊＊公認会計士，システム監査技術者，青山学院大学会計プロフェッション研究科客員教授
〈本稿は，中央大学法科大学院専任教員の推薦を受け，複数のレフェリーの審査を経てここに掲載したものである。〉

され，活用されている。このように，ソフトウェアは，それを利用する企業の業務内容に応じて緻密かつ正確な製品を開発するニーズが増加している。

ところが，企業のソフトウェアへの依存度が高まるにつれソフトウェア開発業務は専門化し，企業は社内での開発から外部のシステム開発会社に開発を依頼するようになった。開発すべきソフトウェアのニーズが，パッケージソフトウェア[2)]から企業の専門的業務に特化した製品まで多様化・複雑化していることから，ソフトウェアの開発は，専門的知識を有する外部の開発業者に開発を依頼する方が，企業の経済的合理性に適うからである。

このように，企業が社内でソフトウェア開発の品質管理をすることが難しくなった結果，開発業者との間で様々なトラブルが発生するようになった。すなわち，ソフトウェアの品質に最終的な責任を負うのは発注元の企業（ユーザー）である一方で，ユーザーはソフトウェアの開発を外部に依存する傾向が強くなるに従って社内でソフトウェア開発に精通した人材が欠乏し，ソフトウェアに関するノウハウが社内に溜まらない状況になったことから，ソフトウェアの品質の実現は外部の開発業者（ベンダー）に依存せざるを得なくなったのである。その結果，企業は開発の最終段階である検収試験の実施までの間は，ベンダーからヒアリングや報告を受ける以外，品質の実現について受け身にならざるを得ず，実際に納品された目的物の内容を巡って，ベンダーとユーザーが対立する事態が後を絶たない。後述するように，ベンダーが行うソフトウェアの品質管理は，完成するソフトウェアの品質に大きな影響を及ぼすものであるから，ベンダーの管理はソフトウェア開発にとって最も重要な課題である。

本稿では，まず，システム監査の観点からのソフトウェア品質管理の考え方を整理する。システム監査は目的によって様々な評価・検証が行われるが，特にソフトウェア開発過程の監査において，システム監査人が留意する観点からソフトウェアの品質管理を検討する。システム監査は，監査対象から独立した第三者であるシステム監査人が，システムに関する業務を検証することによって行われるため，システム監査の視点からソフトウェア開発過程を検討することによって，客観的な観点からソフトウェアの品質管理について整理することが可能となる。

次に，ソフトウェア開発に関する裁判例における「完成」の意義について検討する。そもそも，ソフトウェア開発の法的性質は，後述の通り請負契約と解されるべき場合がほとんどであるが，その特殊性からソフトウェア開発訴訟では様々な論点が存在する。問題となる論点の区分の仕方は様々であるものの，①開発すべき製品の仕様確定，②目的物の完成の判断，③当事者の協力義務，④完成後の瑕疵担保責任の４つに整理するこ

とが可能である。

まず，①開発すべき製品の仕様確定の問題は，請負契約における契約内容の確定作業であり，事実認定の問題である。すなわち，ソフトウェアの開発実務において契約当初の要件定義書の作成の段階では，最終的な製品の仕様は確定しておらず，ベンダーは，ユーザーの専門的業務を踏まえた上で適切なヒアリングを行って最終的な仕様を確定していく必要があることに加え，一度製品の仕様が確定した後も，ユーザーの要望によってその内容が変更・追加されることが頻繁に行われる。また，ユーザーはソフトウェア開発に関する専門的知識を欠いている場合が多く，その上，ソフトウェアはそれ自体が可視性のない無体物であることから，ユーザーは実際にシステムを利用して初めて想定していた製品と完成品が異なるものであることに気付くこともある。このように，ソフトウェア開発では，そもそもベンダーが作成すべきであった目的物の内容を確定すること自体が困難であり，この内容を巡ってユーザーとベンダーで対立するケースが多い[3)]。実際の訴訟においても契約内容の確定は，契約書，要件定義書，詳細設計書，プログラム仕様書，議事録，メール，双方の開発担当者の証人尋問等を実施して，請負契約の完成品としてあるべき姿を確定していくという作業が必要となる。

次に，②目的物の完成の判断は，ソフトウェア開発訴訟において最も重要な論点の一つである。そもそも，ソフトウェアに不具合（一般にバグと呼ばれる）が存在することは不可避であることから，不具合の存在をもって目的物が完成していないと判断されるべきではない。その一方で，製品の品質はユーザーにとって最も重要な関心事であり，業務上の使用に耐えない納品物をもって完成と判断されることはユーザーにとって酷であることから，ベンダーがどのような製品を開発し，どのような作業を行った場合に目的物が完成したと判断されるべきかは，当事者にとって重要な問題となる。この点について，請負契約における目的物の「完成」（民法第 632 条）とは評価的な意味合いも含む概念であるから，ソフトウェア開発訴訟においては，どのような場合に目的物が完成したと判断されるべきかについて，様々な問題点が存在する。

また，③当事者の協力義務の問題は，ソフトウェア開発実務の全ての過程について問題となり得るものであり，請負契約におけるベンダーの債務不履行責任の問題ともいい得る。例えば，ベンダーは，ソフトウェア開発の専門家として，最終的な製品の仕様を確定する責任があることから，ユーザーの業務上の使用に耐えない製品を仕様として確定した場合，債務不履行責任の問題が生じ得る。また，ソフトウェアの開発も請負契約である以上，開発期限を徒過した場合，ベンダーは当然に履行遅滞の責任を負うべきである。しかし，一方で，ベンダーはユーザーの専門的な業務内容については素人である

から，ユーザーの協力がなければ，その業務に使用する製品の仕様を確定することはできない場合が多い。そのためユーザーから仕様確定の適切な協力を得られない場合や，ベンダーの開発作業が最終段階に差し掛かった時点でユーザーから大幅な仕様変更の要望を受けた場合には開発が履行期に間に合わないことは当然であり，そのことを説明したベンダーに落ち度はないというべきであるから，そのような場合にベンダーが債務不履行責任を問われるのは妥当でない。このように，ソフトウェア開発においては，当事者の協力義務は開発過程の全てにおいて問題となるところ，当事者は目的物の完成のために相互に協力すべき義務があることから，個々の事案に応じて当事者が負うべき義務は変容し，この義務に違反した内容・程度によって，ベンダーの債務不履行責任の有無が変わってくる[4)]。

④瑕疵担保責任の問題は，目的物が完成した後は，たとえバグがあってもそれが目的物の瑕疵に当たり，かつ，その瑕疵によってユーザーが契約の目的を達成できないと判断されない限り，ユーザーは契約を解除できない（民法第635条）。その一方で，ベンダーはバグが目的物の瑕疵と判断される場合には，無償でそれを修補すべき義務が発生することから，ソフトウェア開発における瑕疵担保責任の有無は実務上問題となるべき点が多い。

本稿では，上記の問題点のうち，②の目的物の完成の有無の問題について，先行判例における完成の判断内容を検討する。

Ⅱ　システム開発プロセスとソフトウェア品質管理[5)]

1．システム開発

⑴　システム開発の全体像

企業においてシステムを使用する際には，ソフトウェアを入手する必要がある。入手するソフトウェアが汎用的なもので足りる場合にはパッケージソフトウェアを使用することが多い[6)]が，該当製品が見当たらない場合には自らソフトウェアを構築する必要がある。ソフトウェアを自社開発する場合には，一般的に企画，開発，保守の順に行われる。

ソフトウェアの開発は，例えばSDLC（System Development Life Cycle）[7)]と呼ばれるソフトウェア開発の方法論などがあるものの，必ずしも従わなければならないというも

のでもない。近年は開発の形態も多様化していることから，その形態によっても開発の方法論は異ならざるを得ない。開発の方法論については，一義的に定められているものではなく，また確立された唯一のものも存在しない。そのため企業ごとに異なる開発方法論を有していることは一般的であり，それぞれの方法論に従った開発業務が行われることになる。開発方法論は，ソフトウェアの開発形態や使用する開発ツール，プロジェクト体制などによって異なるものと考えられる。しかしながら，開発主体によっては定められた開発方法論を有していないことがある。その場合には一定の開発プロセスにおけるコントロールが有効に機能しない可能性が想定されるため，ソフトウェアに不具合が生じる可能性が高くなるだろう。

ソフトウェアの開発手順が整備されていることによって，その過程で実施すべき項目及び作成されるべき文書等が定義される。それらに準拠することによって，オンプロセスでの作業品質の確保，事後的な検証がともに可能となり，ソフトウェアの品質向上に資するのである。

ソフトウェア開発におけるステップは，概ね次のような項目で整理することができる[8)]。

a） 企 画

どのようなソフトウェアをどのような方法で調達するのかを決定する。一般に，多くの調達要求が各部門から提出されており，企業戦略等との整合性を勘案したうえで，割り当てられた予算の範囲内で調達すべきソフトウェアを決定する。調達を決定する過程においてパッケージソフトウェアによることができるのか等，調達の方法についても議論される。

b） 開 発

個別のソフトウェアに関するプログラムの作成を狭義の開発としてとらえることができる。ここでは調達が決定されたソフトウェアについて，プログラムを作成する。開発を行うに当たっては，要件定義，プログラミング，テストなどの段階を踏むことになる。

c） 保 守

開発が終了し，実際に運用がスタートした後で発見された不具合の改善等によりプログラムの修正を行うことがある。これは開発業務の後工程として行われる場合と，開発終了後の新たな業務として行われる場合がある。

⑵　開発業務のステップ

開発業務はソフトウェアの作成を主に行うため，ソフトウェアの品質を確保する段階としては最も重要な要素である。本稿では，開発業務を要件定義，基本設計，詳細設計，プログラミング，テストの各ステップで整理する。なお，開発のステップや用語については一義的に定められていない。

ソフトウェア開発の形態として，一般にウォーターフォールモデルやスパイラルモデルといった表現で論じられることがある[9)]。開発の形態は，ベンダーとユーザーの関わり合いの違いを表すととらえることもできる。しかしながら，今現在において完全なるウォーターフォールモデルで開発を実施するケースは減少傾向にあると言われている一方で，スパイラルモデルにより開発が行われているかといえば，必ずしもそうではないことが多い。すなわち，ベンダーとユーザーとの間で適時にコミュニケーションをとりつつ，ベンダーがプログラムを作成しているケースが多いのではないかと考えられる。そのため，従来型のシステム開発に関する考え方が必ずしも現在の業務に整合しないことにより，ソフトウェアの不具合が露呈した段階で適切な対応を取り難い事象が発生している側面もあると考えられるのである[10)]。

そのような前提に立ってプログラム作成の方法を検討することになるが，ソフトウェアの品質管理を考慮したソフトウェアの開発ステップとテストとの関係については，**図1**にＶモデルとしてその概要を整理した。要件定義からプログラム作成，テストに至る開発工程のどの段階でどの水準の品質を考慮すべきであるのかが一覧できることが特徴である[11)]。

ここで，ソフトウェア開発における各ステップでの役割を簡単に整理する[12)]。

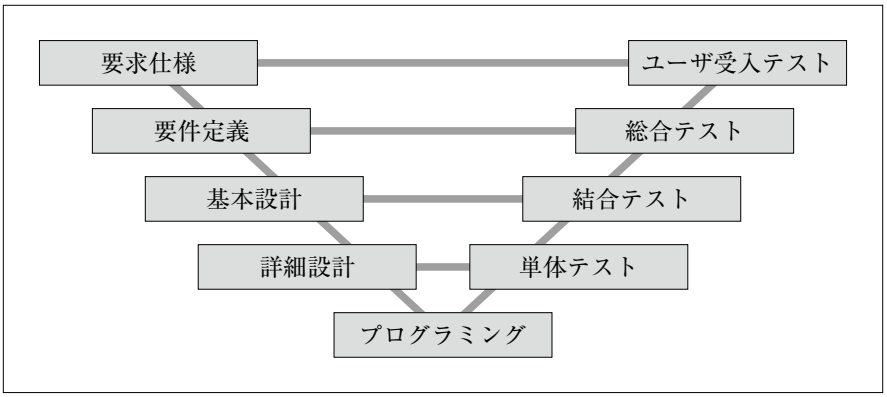
ａ）　要 件 定 義

開発するソフトウェアが，どのような機能を実装する必要があるのか，どの程度の性能が必要なのか，ユーザーのニーズを収集して明らかにしたうえで，業務での利用を考慮しながら優先順位を定め，実装する機能を定義する。ユーザーからのニーズは要求仕様として明らかにすることもあり，そのような場合には要求仕様をベースに実装すべき機能を検討することもある。

ｂ）　基 本 設 計

要件定義にもとづいて，機能レベルで設計する。この段階でプログラムの構成や仕様が決定される。画面の構成や出力帳票，基本的な操作方法などがこの段階で具体化される。入出力項目についても整理することから，データベースの基本構造など基礎

図１　システム開発のＶモデル



的な仕様について決定される。

ｃ）　詳　細　設　計

個別具体的なプログラムを設計する。基本設計で決定した各機能から個別のプログラムレベルに落とし込むことによって，どのようなプログラムを作成するのかを決定する。また，作成するプログラムとして具体的にどのような処理を行うのかについて，プログラムのステップごとに決定する。

ｄ）　プログラミング

詳細設計に基づいてプログラムを作成する。プログラムの作成は，開発ツールと呼ばれるソフトウェアを用いることが多い。プログラムの記述方法については，個人に依存している場合もあるが，基本的なルールを設定して，設定したルールに従って記述することが望まれる。記述方法を統一することによって，不具合を検証する場合に効率性が高まり，またプログラムの修正が事後的に必要になった場合にも高い効率性が期待できる。

ｅ）　テ　ス　ト

テストは，単体テスト，結合テスト，総合テスト，ユーザー受入テストに分類される。それぞれのテストが，要求仕様から詳細設計に至るステップと対応関係を有していることが特徴的である。これらテストの内容については後述するが，テストの各段階において，どのような水準を満たせばよいのか，対応しているステップから検討することができる。例えば，単体テストは詳細設計と対応されるため，個別のプログラムレベルでのテストが必要であることが理解できるのである。

なお，近年ではソフトウェアの開発を効率化するために，従来のような手順とは異なる開発手法が用いられることも多くなってきている。特に，開発ツールが高度化したこ

とに伴い，ユーザーとベンダーがインタラクティブなコミュニケーションを進めることによって，相互にソフトウェアを作り上げていくような開発手法がとられることもある。そのような場合，要件定義にしたがった各種設計書の作成を合理的に簡略化することもある[13]。

2．システム開発の体制（外部委託による開発の管理）

システム開発を論じる際の前提として，企業内に情報システム部が存在し，当該システム部において開発業務を実施するケースを想定する場合が多い。しかしながら今日では，比較的大きな企業においても開発業務を担う情報システム部を設置せず，情報システムの企画部署を設置するのみとしているケースが比較的多くなってきていると考えられる。これには様々な要因があるが，前述のとおりパッケージの利用が進んだことと，継続的な開発案件を抱える企業はそれほど多くなく，その場合には個別に外部に発注した方がコストも含めて効率的であるという判断があると想定される。

そのようななかで，システム開発の管理責任は依頼者であるユーザー側に帰属することになるが，果たして管理するために必要なスキルがユーザー側に常に備わっているかといえば，決してそのようなことはない。企業内で情報システムに関するノウハウを蓄積する場が失われていることもあって，企業において開発管理能力を有する人材が育たなくなっている実情があり，また外部から人材を調達するとしても，継続的に開発案件が存在しないことがそれを難しくすると考えられる。したがって，開発時にはユーザーである企業側から開発プロジェクトの統括者１名に加えて，関係する部門の担当者が数名指名されてプロジェクトに参画している例が多い。ベンダーはこのようなユーザー側の体制を前提にしたシステム開発を行うことになる。

そのため，開発管理を行うに当たってもベンダーの支援は非常に重要であり，ベンダーはユーザーが開発管理を行うために必要な情報を提供する義務を有するほかに，ユーザー側に欠如しているスキルが存在する場合にはそれを積極的に支援しなければ，開発業務を完遂することが困難であるということも含めて，ベンダーにも一定の責任が課されていると考えられる[14]。

それではユーザー側は，どのように開発管理を実施することができるのだろうか。ユーザー側が能動的に開発業務に関する情報を取得することは困難であることが多いため，定期的に開発側からの情報の提供を受けることが必要となる。ベンダーからユーザーへの情報の伝達方法には多くの手段があると考えられるが，一定の書式を双方合意

のもとに決定したうえで，その書式に基づいて定期的に報告するパターンが比較的多い。しかし，ユーザー側が開発情報の能動的な取得が困難であるとはいっても，ただ受け身で情報を待っていればよいわけではないことに注意が必要である。例えば，報告された情報に疑義がある場合や，何か別の原因で異常を察知した場合には開発側に情報の提出を求め，場合によっては行動を伴って現状把握に努めるべきである。これによりユーザー側は継続的なモニタリングを行うことが実効性を有することになるとともに，開発側も情報の作成を定型化することが可能になる。しかしながら，開発業務を進めるにあたって，イレギュラーな事象も当然発生する。そのような場合にはフリーワードで記述することにより，情報の伝達が行われる。

場合によっては，企画段階からベンダーが支援することも考えられる。例えば，ユーザーが自らの要求を整理し，どのようなソフトウェアを構築するのかをベンダーに伝えるために要求仕様書（RFP[15]）と呼ばれるものを作成するが，当該 RFP の作成をベンダーが支援するケースである。RFP を作成する場合には，現在利用しているシステムの概要や業務での利用状況などについて，フローチャートなどを用いることによって整理するとともに，現在のシステムでは実現できない業務や一層の業務改善につなげるために必要な機能等，新しいソフトウェア開発の際に実装してほしいとユーザーが考える要件を洗い出し，新たな業務フローを検討したうえでソフトウェアへの要求を定義する。これらは一般にユーザー側で実施すべきとされている項目ではあるものの，実態としてベンダーが支援しなければならないことも多いのが実情である。

ソフトウェアの開発が終了し，最終的にユーザー受入テストを実施する場合にも，ベンダーの支援が必要なことがある。ユーザー受入テストの実施方法も多様であるが，ユーザー側での実施能力が欠如している場合，ベンダーとしてはユーザーによる品質の確認と検収の必要性から，本来ユーザーが実施すべき受入テストについても支援しなければならないケースがある。

しかしながら，ユーザー側で企画立案し，ソフトウェアの開発業務をベンダーが行い，最終的な品質の確認（検収業務）をユーザーが実施するということで整理すれば，委託側であるユーザーと受託側であるベンダーとの間での役割は概ね**表 1** のように整理できる[16]。

3．ソフトウェア品質管理

ソフトウェアの品質の判断は，最終的には作成目的どおりに利用できるか否かの判断

表 1

ユーザー	ベンダー	内 容
企画立案		
要求仕様書（RFP）作成		新しい業務フローの整理を含む
モニタリング	プログラミング	ユーザーはベンダーの状況を把握する必要がある ベンダーはユーザーに適時に情報を提供する必要がある
モニタリング	テスト	追加開発依頼に対する調整
ユーザー受入テストの実施	ユーザー受入テストの支援	
結果確認・検収	修正・調整	業務で利用できるかの判断 業務で使用できるか否か判断するためのリリース判定会議を開催し，性能を満たすと考えられる場合には検収完了

である[17]。ユーザー受入テストの結果，仮に業務上の利用に耐えうると判断された結果としてリリース判定されたものの，結果的にリリース後に不具合が発見されることがある。このような状況は，テストを実施した際に想定すべきテスト項目が漏れていた場合や，そもそも当該事象を想定できなかった場合などに発生することがある。

ソフトウェアは複数のプログラムによって構成され，プログラムは人が作成するものであることから，全く不具合の無い完全なソフトウェアは存在しないとされている。作成されたソフトウェアは何らかのオペレーティングシステムの上で動作し，またソフトウェアが動作するために必要なライブラリ等と呼ばれるファイルは，開発ツールに依存して提供されるものである場合も多く，構築したソフトウェアが動作するコンピューターシステムの中で多種多様なプログラムが相互に関連しながら動作しており，それらが直接的・間接的に構築したソフトウェアに影響を及ぼすことによって不具合が発生する可能性も否定できない。

このような状況を鑑みれば，ユーザー受入テストを経た後，リリース判定会議において業務上の利用に耐えうると判断した結果，事後的に問題が発覚することを撲滅しない限りソフトウェアの品質が良いと言えないというわけではない。すなわち，ユーザーとしては，一般的に業務上利用できるだけの品質が確保されているか否かが重要な判断基準であり，最終的にはベンダーの意見も踏まえたうえでユーザーが総合的に検討するこ

とによって判断されるべきものである。

4．発生するトラブルの例

しかしながら，最終的にベンダーが納品しなければならないソフトウェアについて，ユーザーとベンダーが対立するケースが見受けられる。その多くは，ベンダーは完成していると主張する一方で，ユーザー側が検収に値しないと判断し，相互が歩み寄れないことによって発生すると考えられる。また，作業スケジュールが遅延することも大きな理由の一つになろう。ソフトウェアの開発はスケジュールが遅れがちになることが多いとみられることがあるが，それは開発業務の進め方に起因するケースが多い。すなわち，実現すべき要件を決定してソフトウェア開発を行うとしても，プログラムの作成を進める段階でミーティング等を重ねることにより，当初要件の修正や新しい要件の追加などが発生する場合が多く，それらを調整し実装することにより，スケジュールとのかい離が発生する。他にも，開発途中に発見された不具合の改修に想定外の時間を要することもあるかもしれない。

このような場合にソフトウェアの開発をすぐに終了してしまうことは，両者にとって望ましいことではない。開発側は収益を得る機会を失ってしまうことになるし，ユーザー側もソフトウェアを利用することによって得られるであろう利益を逸してしまうことになる可能性が高いからである。

そのため，例えば次のような条件を設けることによって，将来に向けて相互にメリットのある解決法を模索することになる。

○　問題となっている不具合の内容を特定し，その不具合の解決について合意する。
○　ベンダーのソフトウェア品質管理体制を見直し，人材を再配置する。
○　不具合対応のスケジュールを明確化し，不具合解消の進捗を管理する。

このような条件を，一定のマイルストーンと併せて設定することによって，当初設定していた期限を超えてしまうことはあるものの，相互に建設的な解決に向けて動き始めることができる。仕切り直しによる再スケジュールにより，当初予定よりもソフトウェアの開発は遅延することになるが，結果として相互の利益喪失が最小になるように相互に努力することが求められる。

なお，開発業務を遂行している過程だけではなく，場合によってはユーザーの検収後に発覚した不具合がトラブルの原因になることもある。この場合，ユーザーにおける検収基準に問題が存在していた可能性が疑われる。また，ユーザー受入テスト時に判明で

きなかった事象がトラブルの原因となっている可能性もある。いずれにしても，このような場合においてはユーザー側とベンダーの双方で原因を調査したうえで相互に協議して対応を進めるほかはない。

5．ソフトウェアの品質の判断

ソフトウェアの品質について最終的に判断するのは発注者であるユーザーであるが，それまでにも品質を確保するためのステップは多くの段階で設定されている。**図1**におけるシステム開発のＶモデルによれば，開発業務におけるソフトウェアテストの位置づけは次のように整理されている[18)]。

○　単体テスト……プログラムを作成する過程で，作成しているプログラムが想定通りに動作することを確認する。主にプログラム作成者によって実施されることが多い。

○　結合テスト……単体テストをクリアした複数のプログラムによって，特定の機能が有効に動作することを確認する。主にプログラム作成者以外の者によって行われることが多い。

○　総合テスト……システム全体として業務上利用可能か否かについて総合的に判断するために行われる。結合テストをクリアしたプログラムの集合として検証されることになり，プログラム作成者とは異なる第三者により行われる。

○　ユーザー受入テスト……ベンダーでシステムテストをクリアしたソフトウェアに対して，ユーザーの立場で業務上利用可能か否かの検証を行う。

ソフトウェアを開発する過程で，このような多段階のテストを経ることには重要な意義がある[19)]。単体テストで検出される不具合は，個別には比較的容易に改修できる場合が多い。例えば，入力されるべきでない値が入力可能となっているケースや，出力情報の形式が異なるケースなどである。個別のプログラムレベルで検出可能な不具合をこの時点で解決しておくことによって，他の段階での不具合検出時点で個別のプログラムに起因している可能性を相当程度低減することができるからである。

また，結合テストや総合テストにおいても，各段階で検出されるべき不具合については，可能な限り解決することが求められる。詳細なテスト手法[20)]に関する議論は行わないが，それぞれのテストの段階ごとに結果の評価を行い，ソフトウェア品質の観点から次の開発段階に進んでよいか否かの判断を行う。このときに条件が付される場合もあり，そのときには次の段階でのテストの際に十分考慮する必要がある。一般に，単体テ

ストはプログラム作成者，結合テスト及び総合テストはテスト担当者が実施し，開発責任者がそれらの結果を確認する必要があると考えられる。各段階での着実な対応が，最終的なソフトウェアの品質に大きな影響を及ぼす。

仮に，単体テストが十分に行われていなかったと仮定した場合，結合テストを実施する段階で発生した不具合の原因が何に起因するものなのか，その分析で考慮すべき事項が多くなる。また個別のプログラムベースでの不具合を解消した結果として，複数のプログラムの集合で支障なく動作するのかをテストする必要があるため，結局はもう一度結合テストに類するテストを実施して不具合を解消しなければ，結合テストを終了してよいレベルのソフトウェア品質であるか否か判断することが難しくなる。

単体テスト及び結合テストを十分に実施しないまま総合テストを実施した場合でも同様の事態が想定される。ベンダーが最終的に納品できると判断可能な品質水準を有するソフトウェアであるかについては，総合テストを実施した段階での不具合の発生状況に基づいて決定されるのである。

なお，ソフトウェアのテストはソフトウェアの品質管理そのものであるが，不具合がゼロとなるケースはさほど多くない現状の中で，どのタイミングで終了すればよいのかが重要な判断となる。当然，検出されている不具合については解消することが基本であるが，例えば，ある一定の条件が揃った場合にのみ発現する可能性のある不具合（得てしてこのような不具合を再現することは難しい）が検出されている場合，その不具合が業務上重要な影響を及ぼさないと考えられるならば，未解決の不具合として整理するものの，最終的にはユーザーによる検収の判断に依存することも考えられる。

総合テストの結果を受けて，ユーザーはユーザー受入テストを実施し，ユーザーの観点から業務での使用に問題はないのかを検証する。多くは通常業務と同一の条件で実施され，旧システムと並行稼働することによって検証を進めることも多い。ユーザーは総合テストまでの結果としてベンダーから根拠データを伴った説明を受けて，ソフトウェアの品質水準はある程度理解している状況にあるが，実際に業務を行う中で問題ないことを確かめることによって検収の可否を決定することになる。ユーザー側では，責任者であるユーザー部門による検収が必要になるものの，ソフトウェア開発に関する専門性の観点から情報システム部門の支援を仰ぐことによって最終的な判断を行うことになる。

ユーザーが最終的な判断を下すにあたっての判断の根拠に定められたものは無いが，整理すると**表 2** のような項目になると考えられる。

表 2

ユーザーによるソフトウェア品質の判断根拠
(i) 最低限の業務をシステムを用いて円滑に実施することができる。
(ii) ベンダーでの開発過程における体制が適切に管理されている。
(iii) 外部委託先であるベンダーの開発状況について依頼者側であるユーザー側で適切に情報を把握できていた（ベンダーからの報告に虚偽がなく，適時適切であった）。
(iv) ソフトウェアのソースプログラムが適切に管理されている。

(i) 最低限の業務をシステムを用いて円滑に実施することができる。

ソフトウェアを開発する目的は，それを用いて業務目的を達成することである。したがって，当該ソフトウェアによって円滑に業務ができなければ目的を達成したとは言えない。ここでは，開発したソフトウェアの使用前と使用後を比較して同程度以上のメリットを得られると判断できる状況が必要である。したがって，ある程度業務上の工夫をすればシステムを用いて何とか業務ができるような状況[21]は，円滑に実施できているとは言い難い。

(ii) ベンダーでの開発過程における体制が適切に管理されている。

開発業務の進捗管理や，各テスト段階における管理など，ソフトウェアの品質を十分に確保できるような体制を敷き，必要な対応を継続的に実施していなければ，一定の品質を保ったソフトウェアを構築できる可能性が低くなる。開発体制の充実度は，ソフトウェアの品質確保には欠かせないものである。

(iii) 外部委託先であるベンダーの開発状況について，依頼者側であるユーザー側で適切に情報を把握できていた（ベンダーからの報告に虚偽がなく，適時適切であった）。

ユーザーによるベンダーの管理も重要な要素である。開発プロセスの進捗管理もさることながら，必要な機能の実装状況などを継続的にモニタリングしつつ，何らかの不具合が発生しそうな場合には，あらかじめ警鐘を鳴らさなければならない。情報収集が完全に受け身であってはならないが，基本的にはベンダーからの適時適切な状況の報告が前提となる。

(iv) ソフトウェアのソースプログラム[22]が適切に管理されている。

ベンダーの最終的な成果物は作成したプログラムである。また開発業務を進める中で，複数回のテストを重ねてプログラムの改修を行っていくことから，複数バージョンのプログラムが生成されることが一般的である。そのような場合に，最新のプログラムの管理状況が明確になっていなければ，常に最新のプログラムによる検証は困難

になるし，結果としてテストを経ないプログラムが納品されるリスクがある。

表 2 の判断根拠のうち，業務で用いることができるか否かがユーザーにとっては最も重要な判断基準となる。したがって**表 2** における(i)が満たされるのであれば，他の項目は判断根拠を構成しないことが多いと考えられる。しかしながら，仮に業務で利用できる品質ではないとユーザーが判断したとしても，その他の項目が全て満たされていた場合には，ユーザーがベンダーに債務不履行責任を認めることは難しいと考えられる。なぜならばソフトウェア開発のプロセスとしては有効に機能していたと判断される可能性が高く，それでもユーザーが要求する品質水準を保持できなかったのは，他の外部要因に起因する可能性が生じるからである。

ソフトウェアには何らかの不具合が残っていることが多く，SDLC のなかで比較的時間をかけながら成熟させていくことが不可欠であるし，ユーザー受入テストの時点で未だ発見されていない不具合が内在している可能性も否定できない。そのためソフトウェアの検収の可否の判断として発見されている不具合の数や割合だけを根拠とするのは，おそらく実態に合わない。通常は特にベンダーで発見された不具合とその内容，解消の状況などを管理したうえで，相対的な判断が行われる。したがって，適切なタイミングで必要な不具合の解消を行い，最終的に品質がある水準以上になっていると判断するためには，ソフトウェアの製作過程における管理状況が重要な判断指針になるものと考えられる。そして，その判断を行うために必要な情報が，責任者に対して十分に伝えられていなければならないのである。

Ⅲ　裁判例における完成の判断

1．ソフトウェア開発契約の法的性質

ソフトウェア開発契約の法的性質について，裁判例[23)]はベンダーがシステムの完成義務を負っているか否かによって請負契約か否かを判断しているところ，ユーザーは業務に使用するために多額の金銭を払ってベンダーに開発を依頼している以上，ベンダーはシステムの完成義務を負っていると解されるべき場合がほとんどであり，ソフトウェア開発契約の法的性質は請負契約である。

したがって，ベンダーは，請負契約に基づいてソフトウェア開発における仕事を完成すべき債務がある。

そして，ソフトウェア開発契約は請負契約である以上，ベンダーは目的物が完成すればユーザーに対する報酬請求権が発生し，その後は開発したソフトウェアの不具合について瑕疵担保責任の有無のみが問題になるのに対し，仕事が完成していなければ，請負人の報酬請求権の発生は後履行であることから，いつまで経ってもベンダーの報酬請求権は発生せず，逆にユーザーから債務不履行責任が追及されることになる。

また，ソフトウェア開発においては，ユーザーは多くのコストを掛けてベンダーに開発を依頼している以上，業務上の使用に耐えない目的物を納品されたことをもって，請負人の仕事が完成したと判断され，ベンダーに対して１年間しか瑕疵担保責任を追及できないとされてしまうことは非常に酷である。特に，ユーザーは新システム導入の必要性からベンダーの責任を追及する前に他のベンダーにシステムの開発を依頼し，その完成後に訴訟提起を検討することも多い。そして，ソフトウェア開発は建物建築等の他の請負契約の場合と異なり，あるベンダーが開発したシステムを他のベンダーが引き継いで完成させることはほとんど皆無であり，ベンダーを変更する場合には一から開発作業を行うことになるので，完成までに時間が掛ることが多い。また，ユーザーはソフトウェアの専門的知識を欠いていることが多く，ベンダーの協力が得られなければ，不具合内容の分析が困難であること等から，ベンダーに対する責任追及まで時間が掛るケースが多い。それにもかかわらず，ソフトウェア開発における目的物が完成したと容易に判断されてしまうと，ユーザーの瑕疵担保責任の追及が既に時効に掛ってしまっている場合がある。

一方，ベンダーにとっても，開発には多くの人員とコストを要するものであるところ，ユーザーの完成判断が適切になされないことはベンダー企業にとって死活問題となり得る。実際の開発実務でも，契約当初に開発費用のみを受け取り，報酬は目的物の検収後に支払うとの特約がなされる場合も多いところ，開発の過程におけるユーザーの新規要望や予想外のトラブルによって予想以上のコストが掛ったにもかかわらず，完成後にもユーザーから報酬が支払われないことによって，ベンダー企業の倒産問題に発展する場合もある。

したがって，実務上，ソフトウェア開発における目的物が完成したか否かの判断の区別は極めて重要になる。

２．ソフトウェアの特殊性

ソフトウェアには，以下のような特殊性が存在することから，請負人の仕事の完成の

有無を判断するには，他の請負契約とは異なる考慮が必要になる。

すなわち，ソフトウェアは，典型的な請負契約の対象物とは異なり，納品物それ自体は可視性のない無体物であることから，実際にこれを利用・運用する段階になって初めて多数のバグが発覚する場合も多い。特に，ソフトウェア開発は多くのプログラムを組み合わせて完成させる複雑システムの構築であるにもかかわらず，請負契約の典型である建物建築請負の設計図面等のように行政法規によって作成すべき書類が整備されていないことから，実物と図面を比較することによってその開発の適否を判断することが困難である。

また，ソフトウェアは，その性質上，一切のバグが発生しないということは想定されず，ベンダーはユーザーに目的物を引き渡した後もバグの修正作業をすることが当然に予定されているというべきである。したがって，バグが発生していることだけをもって目的物が完成していないと判断されるべきではない。

したがって，ソフトウェア開発における請負人の債務については，これらの特殊性を踏まえた上で，仕事の「完成」の有無を判断すべき必要がある。

3．近時の裁判例について

そこで，ベンダーがどのような作業を行ったときに，「仕事を完成」（民法第 632 条）させたといえるかについて，近時の裁判例を検討[24)]する。

(1) ユーザーによるシステムの本番稼働がなされている場合

まず，裁判例①東京地裁平成 14 年 4 月 22 日判決（判タ 1127 号 161 頁）は，被告から販売管理，製造，会計等のシステム開発を請け負った原告が，仕事が完成しているにもかかわらず報酬が支払われないとして，被告に対して請負契約に基づく報酬請求等をした事案である。判決は，「請負人が仕事を完成させたか否かについては，仕事が当初の請負契約で予定していた最後の工程まで終えているか否かを基準として判断すべきであり，注文者は，請負人が仕事の最後の工程まで終え目的物を引き渡したときには，単に，仕事の目的物に瑕疵があるというだけの理由で請負代金の支払を拒むことはできないものと解するのが相当である。」とし，原告の開発した新システムが被告の元で本番稼働されたことを重視して，そのような場合には請負人は仕事を最後の工程まで終えており，原告の仕事は完成していると判断した（なお，原告の納品した新システムは，本番稼働後にも多数の重大な不具合が発生し，稼働後約 1 年後には被告が新システムの利用を断念し，被

告が元々使用していた旧システムに戻さざるをえなかったことから，原告の瑕疵担保責任は肯定した。）。

また，裁判例②東京地裁平成 22 年 1 月 22 日判決（判例集未登載，LLI/DB 06530093）は，被告から大学の事務システムを刷新するシステム開発を請け負った原告が，仕事が完成しているにもかかわらず報酬が支払われないとして，被告に対して請負契約に基づく報酬請求等をした事案である。判決は，仕事の完成について裁判例①と同様の基準を採用し，さらに，ベンダーの仕事の完成を指す請負人の仕事の最終の工程について，「本番稼働を開始するまでの作業が最終の工程であり，本番稼働後の作業は本番稼働前の仕事の成果物の不備を補修する別個の債務の履行であるものと認められる。」とし，ベンダーの仕事はシステムが本番稼働すれば完成することを明示した（なお，原告の開発したシステムが本番稼働後 1 年経過した後もバグの発生が収まらなかったことから，原告の開発したシステムの「バグの発生量は通常のプログラム開発において想定される範囲を超えていることは明らかである。」として，原告の瑕疵担保責任は肯定した。）。

このように，裁判例①及び②は，ベンダーが提供したシステムがユーザーの業務上の使用に耐えないものであっても，ユーザーが提供されたソフトウェアを本番稼働している場合には，ベンダーの請負契約における仕事は完成していると判断した。

⑵　ユーザーによるシステムの本番稼働がなされていない場合

これに対し，ユーザーによるソフトウェアの本番稼働がなされていない場合には，以下のような裁判例がある。

裁判例③東京地裁平成 16 年 12 月 22 日判決（判タ 1194 号 171 頁，判例時報 1905 号 94 頁）は，原告から販売管理システムの開発を請け負った被告が，システムを開発して原告に納品したものの，その納品物に重大な不具合があるとして，原告が被告に対して瑕疵担保責任ないし債務不履行責任に基づく契約の解除及び損害賠償請求等をした事案である。判決は，被告が納品した販売管理システムの不具合について，「契約の目的を達成することのできない重大な瑕疵に該当するというべきである。」としながらも，当事者が作成した契約書の条項の解釈として，被告から原告に対して現実にシステムが交付されれば，それをもって被告は債務を履行したことになる旨の合意があったと認定し，被告の開発したシステムに上記の通り重大な不具合が発生していることを認めつつも，被告から原告に現実にシステムの交付がなされている以上，債務不履行責任の問題は発生しないと判断した（なお，上記不具合の発生について，被告の瑕疵担保責任は肯定した。）。

裁判例④東京地裁平成 17 年 4 月 14 日判決（判例集未登載，LLI/DB 06031541）は，被告から総合物品管理システムの開発を請け負った原告が，仕事が完成しているにもかかわらず報酬が支払われないとして，被告に対して請負契約に基づく報酬請求等をした事案である。判決は，原告の被告に対する報告書の中に，システムの完成度が当事者の合意した本番稼働予定日である平成 14 年 12 月 2 日の直前の同年 11 月 27 日の時点で 60％～ 70％であり，正式な本番稼働が平成 15 年 2 月になる旨の記載があったことから，システムの納品自体はなされているものの完成には至っていないとして，請負人の仕事の完成を否定した。

裁判例⑤東京地裁平成 21 年 7 月 31 日判決（判例集未登載，2009 WLJPCA 07318003）は，被告が原告から新情報システムの開発を請け負って納品したところ，原告は，被告が開発したシステムが不十分であるとして，被告に対して債務不履行ないし瑕疵担保責任に基づく損害賠償請求等をした事案である。判決は，原告が被告の納品したシステムの検収を終えているものの，当事者の契約内容として，被告が納品物である新情報システムの納品前に単体試験及び結合試験[25]を実施し，その結果をテスト結果報告書という書面で原告に提出することが合意されていたところ，納品されたシステムに，被告がきちんと単体試験及び結合試験を実施していれば発生しないであろう多数の不具合が発生したことから，被告は単体試験及び結合試験が未実施であり，かつ，テスト結果報告書の未提出という債務不履行があるとして，原告の契約の解除及び損害賠償請求を認めた。

裁判例⑥東京地裁平成 23 年 4 月 6 日判決（判例集未登載，LLI/DB 06630209）は，被告が原告から請け負った歯科医院向けのレセプトコンピューターを開発したところ，原告は，被告が作成したレセプトコンピューターには多数の不具合が発生し，原告の検収が不合格になったとして，原告が被告に対して支払った請負代金の返還請求等を求めた事案である。判決は，当事者の検収試験の結果や不具合の分析等をもとに，被告の納品物が「仕様書において合意された機能の大部分が実装されていなかったり，正しいデータや必要なデータが出力されなかったり，実用に耐える性能を有していないなど（以下，これらの機能の不備，不具合等を「本件不具合等」という。），客観的に未完成である。」との原告の主張をそのまま認定し，仕事の完成を否定した。

⑶　裁判例における完成の判断

ベンダーが納品したソフトウェアがユーザーのもとで本番稼働されていない裁判例③ないし⑥では，裁判例③を除き，ベンダーが納品したシステムの不具合の内容を具

体的に検討した上で，納品物が当事者間の契約上の仕様を満たしているか，換言すれば，ユーザーの業務上の使用に耐えうるかという観点から請負契約における仕事の完成の有無を判断している。これは，本番稼働がなされたかという行為の外形から仕事の完成の有無を判断している裁判例①及び②とは明らかに異なる基準を示していると解釈できる。なお，裁判例③は，システムの現実の交付によってベンダーの請負契約における仕事の完成を認めていることから，一見すると，裁判例①及び②に類似しているようにも思えるが，裁判例③は契約条項の解釈という当事者の意思解釈の問題として現実の交付をもって仕事の完成と認定しているのであるから，他の裁判例と別異に解することはできる。ただし，システム開発ではユーザーは多額の開発費用及び時間を費やしてベンダーに業務を依頼していることから，どのようなソフトウェアでも現実の交付さえなされれば，ベンダーの請負契約における仕事は完成していると契約条項を解釈したことには疑問が残る。そもそも，請負契約における請負人の債務の本質は，仕事の完成義務であるところ，請負人であるベンダーがユーザーの業務上のニーズに基づいて開発を依頼されているのであるから，実際にユーザーの業務上の使用に耐えうるものを完成させることが，請負契約におけるベンダーの仕事の完成と捉えるべきである。

また，裁判例⑤は，ベンダーが単体試験や結合試験を適切に行っていればおよそ発生しないであろう不具合が発生したこと及びテスト結果の報告書の未提出という債務不履行から仕事の完成を否定したものである。これは，不具合の内容が，通常のシステム開発の過程で行うべき検証作業を行っていればおよそ発生していないであろうバグが発生していること等からベンダーが単体試験や結合試験を行っていないと判断したものであり，ベンダーから納品されたソフトウェアの不具合の内容を検討した上でベンダーの債務不履行責任を肯定している点においては，他の裁判例と矛盾するものではないと考えられる。

しかし，裁判例⑤は，ベンダーが契約内容である単体試験や結合試験の結果を納品せず，かつ，納品したシステムの不具合の内容が単体試験や結合試験を適切に実施していればおよそ発生していないであろうものであったことから，ベンダーが単体試験や結合試験を実施していないという，開発過程に着目した点では他の裁判例とは異なる。換言すると，不具合の内容から，ベンダーが単体試験や結合試験の実施していない（又は適切に実施できなかった）というベンダーの品質管理の欠如を認定したとも評価できるからである。

このように，ベンダーの品質管理能力に着目することは，ソフトウェア開発が途中で終了してしまったケース（ベンダーから提供された納品物がユーザーの検収試験[26]に不合格

となり，その後の開発が中止となったことから，ユーザーがベンダーに対して債務不履行に基づく契約の解除を主張し，これに対して，ベンダーがバグは軽微であり，容易かつ短期間に修補が可能であることからベンダーに債務不履行はないとして，逆に請負代金の請求をしてくるというケース）のように，両者の信頼関係が破壊されてしまった場合に，判断の重要な一資料になると思われる。

すなわち，ソフトウェア開発では納品物に不具合があった場合でも，ベンダーがこれに遅滞なく対応できている場合には，ベンダーにその責任は発生しないと考えられている[27)]。

そして，ベンダーの対応の可否を検討するには，ソフトウェアのバグの内容及びベンダーの修補能力の把握が必要になるところ，開発が途中で終了した以上，ユーザーがこの問題点について専門的な判断を下すことはおよそ不可能であり，ベンダーによる不具合の原因調査の協力が不可欠になる。例えば，バグの内容が，システム開発がウォーターフォール方式でなされ，かつ，発生したバグを修正するのにシステムの外部設計[28)]からやり直さなければ不具合を修正できないような深刻なバグが発生している場合から，ベンダーがソフトウェアの不具合を容易に修正でき，かつ，バグを修補しても他の部分に影響しないといえるような軽微なバグしか発生していない場合まで様々なものが想定されるが，特に，ベンダーがどのような方式でソフトウェアを開発しており，発生したバグや仕様漏れとの関係で，ベンダーがどのような作業工程から補修作業をやり直すことが必要になるかは，実際にソフトウェアを開発したベンダーにしか分からないものである。また，発生した不具合を容易に補修できるかどうかは，ベンダーの規模や人員等の主観的な補修能力に関わるものであるから，客観的な基準の定立は不可能といっても過言ではない。

もっとも，ベンダーが遅滞なくバグを対応できるか否かは，ベンダーの品質管理能力に直結する問題である。なぜなら，本稿Ⅱに記載の通り，ソフトウェア開発では，要件定義から検収試験に至るまでの開発プロセスが極めて重要であり，ベンダー側だけでも，単体試験，結合試験，及びシステム試験[29)]といった様々な試験を行い，的確なプロジェクト体制を構築する等，ベンダーが適切な品質管理を行うことが当然の前提とされていることから，ベンダーの品質管理能力が高ければ不具合の修補能力も高いと考えられる。

したがって，例えば，ベンダーが納品した目的物から，ベンダーが単体試験や結合試験を適切に行っていればおよそ発生しえない不具合を相当数発生させている場合には，ベンダーは開発過程において，単体試験や結合試験を適切に行うという品質管理を欠い

ていることが推認されることから，この点についてベンダーからの反証がない限り，そのような発生したバグに対してベンダーが遅滞なく対応することは困難であるというべきであろう。

裁判例⑤は，上記問題点を解決するための重要な試金石となりうる裁判例と評価できる。

⑷ 検 討

このように，近時の裁判例[30]は，システム開発におけるベンダーの仕事の完成の有無について，ユーザーが納品されたシステムの本番稼働をしている場合には，たとえ納品されたソフトウェアに重大な不具合が発生している場合であっても，ベンダーの仕事が完成していると，いわば形式的に判断しているのに対し，ソフトウェアが本番稼働に至っていない場合には，ベンダーから納品されたソフトウェアの不具合を具体的に検討した上で，それが当事者の合意した契約上の仕様を満たしているか，換言すれば，システム開発を依頼したユーザーの業務上の使用に耐えうるかどうかという実務的な観点から，実質的にソフトウェアの完成度を検討し，ベンダーの仕事の完成の有無を判断していると解釈できる。

ところで，一般的に，納品されたソフトウェアがユーザーの行う検収試験に合格すれば，ベンダーの請負契約における仕事は完成したと判断するとの考え方が主流である[31]。これは，ソフトウェアの開発は，ユーザーの検収試験をもって運用段階に移行するので，ベンダーの開発業務は終了したと考えることができるからである。そして，ユーザーにおける本番稼働は，通常は検収試験の合格後に行われるものであるから，本番稼働がなされていれば，たとえ検収をしていなくともベンダーの仕事は完成したと判断でき，請負人の業務の最終工程に達したという考え方が，前記裁判例①及び②の背後にあると推測される。もっとも，ソフトウェア開発は，開発に多くの時間とコストを要することから，ベンダーの納品後のユーザーのタイムスケジュールはタイトであり，新システム導入の必要性・緊急性から，いわばユーザーがベンダーに押し切られる形でソフトウェアの検収及び本番稼働に至る場合が多いにもかかわらず，本番稼働をもってベンダーの仕事が完成し，後は瑕疵担保責任しか追及できないというのはユーザーにとって酷な結論ともいえる。

すなわち，ユーザーが適切な検収試験を実施できる程度の製品に対する知識やノウハウがある場合には，ユーザーの検収試験の判断を信用し，検収及び本番稼働がなされていることをもって請負契約における仕事の完成の有無を判断してよいとも思える。しか

し，ユーザーに検収の適否を判断できる知識やノウハウがないことがほとんどである現状に鑑みると，請負契約における仕事の完成の有無に直結する検収試験の合格の判断をユーザーの担当者のみに負担させることは酷である。この点は，ユーザーとベンダーの協力義務にも直結する部分でもあり，ベンダーとユーザーの適切な信頼関係に基づく検収体制が整備されればこの問題は解決されうる問題ともいえる。

しかし，請負契約における請負人の本来的債務は目的物の完成義務にあり，ソフトウェア開発において，ベンダーは，システム開発の専門家であることから，ユーザーの業務上の使用に耐えうるシステムを開発することこそが，ユーザーから依頼された本来的債務というべきである[32)]。したがって，ユーザーの検収試験が適切に行われた場合を除き，ユーザーの検収は形式的であり，実質的な判断が行われていることは稀であると考えざるをえない。また，ソフトウェア自体は可視性のない無体物であり，ユーザーは，適切な検収試験を実施していない場合には，実際にシステムを本番稼働するまでユーザーが納品物の適否を判断できない場合も多いことを考慮すべきである。

したがって，ソフトウェア開発における完成の判断は，検収や本番稼働という外形のみに囚われず，本番稼働がなされていない場合と同様に，実際に納品されたソフトウェアの内容を検討した上で，ユーザーの業務上の使用に耐えうると判断できる場合にのみ，請負人の仕事が完成したと判断されるべきであり，前記裁判例①及び②のように，ベンダーが納品したソフトウェアについて，本番稼働後に多数の不具合が見つかり，ユーザーの業務上の使用に耐えないことが判明した場合でも，本番稼働をもってベンダーの請負契約における仕事が完成したと判断している点は，請負契約の債務の本旨から考えて疑問が残る。

Ⅳ　おわりに

以上で検討した通り，近時の裁判例は，システム開発におけるベンダーの仕事の完成の有無について，ユーザーが納品されたシステムの本番稼働をしている場合には，たとえ納品されたソフトウェアに重大な不具合が発生している場合であっても，ベンダーの仕事の完成を認定している。これに対し，システムが本番稼働に至っていない場合には，ベンダーから納品されたソフトウェアの不具合を具体的に検討した上で，それが当事者の合意した契約上の仕様を満たしているか，換言すれば，システム開発を依頼したユーザーの業務上の使用に耐えうるかどうかという実務的な観点から，実質的にソフトウェ

アの完成度を検討し，ベンダーの仕事の完成の有無を判断していると解釈できる。

しかし，ソフトウェア開発における目的物の完成の判断については，提供された目的物の内容を客観的に判断し，提供された納品物が，ユーザーの業務上の利用に耐えることができるかという観点からなされるべきであり，このことは，ユーザー受入テスト（検収試験）や本番稼働を経たからといって変わらないのではないだろうか。

これらは，ベンダーが目的物の完成義務を負っているという請負契約の法的性質から導かれる結論であり，可視性のないソフトウェアという目的物の性質及びユーザーとベンダーの能力の差が顕著な現代のソフトウェア開発実務を考慮すると，当然にベンダーが負うべき責任であると考えうるからである。

次に，本稿Ⅱで述べたソフトウェア開発プロセスにおける品質管理の考え方は，開発過程におけるベンダーの内部統制[33]に大きく影響されるものであり，一方でソフトウェアの開発形態や最新の技術動向に左右されない普遍的なものである。なぜなら，開発過程におけるベンダーの内部統制が有効に機能することによって，はじめて目的物の品質管理の水準向上を図ることが可能になるからである。

したがって，ベンダーは自らの品質管理の向上に傾注すると共に，裁判上においてもベンダーのソフトウェア開発過程における品質管理能力が十分に考慮されるべきである[34]。具体的には，本稿Ⅲで述べたとおり，ベンダーがソフトウェアに発生したバグに遅滞なく対応できるかという債務不履行責任が問題となる場面において，ベンダーが納品した目的物について，単体試験や結合試験を適切に行っていればおよそ発生しえない不具合を開発の最終段階で相当数発生させている場合には，ベンダーはソフトウェアの開発過程において，単体試験や結合試験を適切に行うという品質管理能力を欠いていること，換言すれば，開発過程での内部統制が有効に機能していなかったことが明らかである。したがって，そのような品質管理能力を欠いたベンダーの下では，発生したソフトウェアのバグに対して遅滞なく対応することは困難であると推認されるべきではないだろうか。

注

1） 会計仕訳は，日付，勘定科目，金額により構成される。コンピューターは大量かつ定型的な処理を高速に行うことができ，また会計仕訳の構成要素はすべて数値で表現できることから，処理対象としての親和性も高いという特徴を有している。そのため，会計業務をコンピューターに依存することは，比較的容易であったと考えられるのである。

2） 従来ソフトウェアは個別の用途に応じて構築されていたが，共通の処理を行う場合には多くのユーザーが利用可能なソフトウェアを構築して一般に販売するほうが効率的であり，ユーザーも

安価に高い機能を利用することができる。そのように販売されるソフトウェアをパッケージソフトウェアという。一般に業務系のパッケージソフトウェアの場合には，コンピューターにインストールしただけでは利用することができず，利用するための各種設定（カスタマイズという場合もある）が必要になることが多い。

3） 東京地裁平成 22 年 9 月 6 日判決（判例集未登載，2010 WLJPCA 09068009）は，ユーザーである被告が原告の納品したシステムが契約における仕様通りになっていないとして，請負代金を支払わないことから，原告が請負契約における報酬請求等をした事案であるが，判決は，原告と被告の契約締結前後のやりとりから契約書の条項とは異なる契約内容を確定し，引き渡された内容でベンダーの仕事は完成していると判断した。

4） 東京地裁平成 16 年 3 月 10 日判決（判タ 1211 号 129 頁）は，ベンダーは注文者であるユーザーのシステム開発へのかかわりについても，適切に管理し，システム開発について専門的知識を有しないユーザーによって開発作業を阻害する行為がなされることのないようにユーザーに働きかける義務（以下，これらの義務を「プロジェクトマネージメント義務」という。）を負っていたというべきであるとして，ベンダーは開発期限までに目的物を完成させるため，システム開発の専門家として適切にプロジェクト管理を行う義務があることを認定している。具体的には，ユーザーに対して，「要求の撤回や追加の委託料の負担，納入期限の延期等を求めるなどすべき義務」があるとしている。その一方で，ユーザーについても，本事案で開発すべき目的物が特殊な電子計算システムであったことから，「本件電算システム開発契約は，いわゆるオーダーメイドのシステム開発契約であるところ，このようなオーダーメイドのシステム開発契約では，受託者のみではシステムを完成できないのであって，委託者が開発過程において内部の意見調整を的確に行って見解を統一した上，どのような機能を要望するかを明確に受託者に伝え，受託者とともに，要望する機能について検討して，最終的に機能を決定し，さらに，画面や帳票を決定し，成果物を検収するなどの役割を分担することが必要である。」などとして，ユーザーは，「必要な協力をベンダーから求められた場合，これに応じて必要な協力を行うべき契約上の義務（以下「協力義務」という。）を負っていたというべきである」として，ユーザーの協力義務も認定した。このように，ベンダーとユーザーはシステム開発の全過程において双方に尽くすべき債務が存在し，これらの義務の履行はベンダーの債務不履行責任の有無に直結する。例えば，東京地裁平成 18 年 6 月 30 日判決（判時 1959 号 73 頁）は，ユーザーからの追加要望の多さがベンダーの履行遅延の原因となったが，ベンダーは開発に進行如何によってはユーザーの要望を拒否すべき義務があるとして，単に変更要望が多かったことをもって遅滞の責任を負わないものではないことを認定している。この判例における請負人のプロジェクトマネージメント義務の詳細な分析として，生田敏康「電算システム開発契約における注文者の協力義務と請負人のプロジェクトマネージメント義務」福岡 52 巻 4 号 471 頁がある。また，本稿では詳細に触れないが，同論文及び同じ著者の「注文者の協力義務―コンピュータソフト開発契約をめぐる最近の判例を中心に」福岡 52 巻 4 号 379 頁では，開発過程におけるユーザーの注意義務について詳細な分析，及びベンダーの救済方法について解説がなされている。また，笠井修「注文者の協力義務」好美清光先生古稀記念『現代契約法の展開』（経済法令研究会，2000 年）では，請負契約における注文者の行為に応じて，注文者の負うべき義務について分析されている。

5） 本稿では，複数のプログラムが有機的に一体となって稼働する状況にあるものを「ソフトウェア」という。また，業務を遂行するために用いるソフトウェア群を「システム」と呼ぶことにする。すなわち，システムは，ソフトウェアによって構成された業務で利用可能な状況にある仕組みを含んだ概念である。

6） パッケージソフトウェアといっても，ソフトウェアを調達後，サーバ等にインストールすることのみによって使用できる場合は極めて少ない。一般にはパッケージソフトウェアにカスタマイズすることによって，企業で利用することが可能な情報システムとして成立する。その過程で企画・開発と同様の手順が必要になる。

7） SDLC はシステム開発ライフサイクルの略称である。ウォーターフォールモデル（後掲注 9 参照）による開発を前提にしていると言われることが多い。ソフトウェア開発において，保守のフェーズまで取り込んだ考え方であり，補修を繰り返すことによってソフトウェアの持つ不具合を解決することができると説く。

8） システム管理基準において，システム開発に関する項目は次のように整理されている。なお，「システム管理基準は，本管理基準と姉妹編をなすシステム監査基準に従って監査を行う場合，原則として，監査人が監査上の判断の尺度として用いるべき基準となる。」とされている。（経済産業省「システム管理基準」前文）

Ⅲ．開発業務
1．開発手順
2．システム設計
3．プログラム設計
4．プログラミング
5．システムテスト・ユーザ受入れテスト

9） 要件定義，基本設計，詳細設計，プログラミングのように，設計から構築までの過程を順に確定することによってソフトウェアを開発する方法をウォーターフォールモデルという。またベンダーがプロトタイプを作成し，それをユーザーが評価することを繰り返すことによってソフトウェアを開発する方法をスパイラルモデルという。近年では，効率的なソフトウェアの開発を目指して，アジャイル開発（開発するソフトウェアを小さなプログラム単位に分割し，そのプログラムの開発とユーザーを交えた検証プロセスの反復を繰り返すことによってソフトウェアを開発する手法）と呼ばれる考え方をベースにした開発方法論が展開されている。

10） 例えば，当初要件定義された項目のみが最終的な要件を構成するわけではなく，開発過程でユーザーの要望を採用することにより随時変更が加えられた結果として，完成物が定義されることもある。このような開発プロセスが両者で合意されていることから，要件定義に記載されていない機能について必ずしも開発義務がベンダーに発生しないということにはならないのである。

また，開発環境の変遷もこのような状況を生み出している。見逃せない要素である。従前はコンピューターが全般的に高価であり，作成するソフトウェアを決定したうえで完成したものをベースに検証してもらう方法以外は取り難かった。しかしコンピューターの性能が急激に向上し，それ自体コモディティ化することによって，ベンダーとユーザーとの間でコミュニケーションをとりながらソフトウェアを作成する手法が現実問題として採用されるようになったこともある。またプログラムの作成といっても，開発ツールの進化により，リッチな開発環境を提供することが可能となっており，いわゆる命令文を記述しなければならないケースは減少している。

11） 例えば，IT が可能にするサービスの業界ベストプラクティスを示した ITIL（IT Infrastructure Library）には，サービス V モデルとして次のような記述がある。「テストのレベルは，システムが設計および構築された方法から導き出される。これは V モデルとして知られ，テストの種類を開発の各段階に対応付けるものである。V モデルは，サービストランジションの各レベルでのテストが，サービス要件と設計の対応する段階としどのように一致するかを示す一つの例である」ITIL「サービストランジション」（TSO，2008 年）123 頁。

12） ここで整理している各ステップは，一義的に定められたものではなく，開発プロジェクトによっても求められる内容は変わってくるし，分類方法も異なることに留意が必要である。

13） 各種ドキュメントの簡略化は，コスト削減には非常に有効である場合が多い。また，小規模なソフトウェア開発であれば，効率性の向上が大きなメリットとして感じられると想定される。しかしながら，ある程度の規模以上のソフトウェア開発を行う場合には，情報共有を含めたコミュニケーションの円滑化や品質管理を行う観点から，開発関連ドキュメントの作成を削減することによって情報共有の困難さや検証可能性の欠如などの欠点が目立つことが多くなると考えられる。

14） 実態としてユーザーだけで開発管理を行うことは困難な場合が多い。詳細については，後掲注

20 参照。

15） RFP（Request For Proposal）は，構築するソフトウェアに対するユーザーの要求を整理したものであり，システムを開発する前提となる資料である。特に複数ベンダーに対する見積り依頼資料として利用される場合も多い。

16） V モデルとの関係は次のように説明可能である。RFP の作成とそれに対応するユーザー受入テストについては，ユーザーの役割として整理される。一方で要件定義に基づいたソフトウェアの設計や，その設計レベルに応じたテストはベンダーの役割として整理されることになる。

17） 監査上の品質保証として，監査の目的は次のように整理することができる。「品質保証活動は計画され文書化される。プロジェクト活動と製品が適切な基準・手続・要求に従うことは，客観的に正しい。影響のあるグループすべてが品質保証活動に気づき，協力する。従われない部分は経営者によって対応される。」（Frederick Gallegos, Sandra Senft, Daniel P. Manson, Carol Gonzales Auditing IT Planning And Organization 206（2004）.

システム管理基準においては，システム設計においてユーザーによる承認が，ユーザー受入テストとしてユーザー及び運用の担当者もテストに参画して確認することが求められている。これが一つの基準であるとするならば，そのような基準等に従って開発業務を遂行することが，目的通りに利用できることを保証するものであり，それが判断基準であると考えられる。

18） 日本公認会計士協会 情報システム委員会編『情報システムのコントロールと監査 Q&A』172 頁（中央経済社，1998 年）では，次のように整理されている。

⑴ 単体テスト

プログラミングをプログラマに指示する最小の単位（プログラム仕様書といったりします）でのテスト。

ここでは，最小の単位での仕様書通りにプログラミングができているかをテストします。

⑵ 結合テスト

一つの意味のある処理となる開始から終了までのデータの動きを追うことによって，仕様書通りのプログラミングができているかをテストします。

⑶ 総合テスト

本番に近い環境でのテスト。

ここでは，本番に近い環境で意図した処理速度で適切な処理結果を得られるかをテストします。

19） テストの区分や分類は，採用する開発方法論や開発環境などによって異なるものであり，一義的に定められているものではない。一定以上の品質を確保するために最も有効なテスト方法を決定することが必要である。

20） テストの方法にはさまざまあるが，例えば，プログラムの処理ロジックに沿って，様々な条件に応じて想定通りに動作することを確かめるホワイトボックステスト，ソフトウェアの詳細な処理ロジックは考慮せずに様々なパターンのインプットを投入しアウトプットを検証するブラックボックステストなどがある。

21） 例えば，業務上必要な情報を出力できないような状況では，円滑な業務遂行には程遠い。入力段階で入力すべき項目に入力できない状況も同様である。

22） 一定のプログラミング言語に基づいて記述したプログラムコードを「ソースプログラム」という。プログラムの修正を行うことは，ソースプログラムの修正を行うことになる。コンパイラと呼ばれる変換ソフトを用いるなどしてソースプログラムから実行可能なプログラムを生成する（ただし，全てのプログラミング言語がコンパイル等の変換を要求するものではなく，インタプリタ言語と呼ばれるようなソースコードに基づいて実行される言語も存在する）。

23） 東京地裁平成 3 年 2 月 22 日判決（判タ 770 号 218 頁）。なお，請負契約以外には準委任契約ないし労働者派遣型の契約類型も考えられる（東京地方裁判所プラクティス委員会第二小委員会「ソフトウェア開発関係訴訟の手引」判タ 1349 号 4 頁）が，ユーザーがベンダーに開発を依頼するシステム開発契約の典型例は請負契約である。なお，準委任契約ないし労働者派遣型契約の場合に

は，目的物が完成しなくとも，その従事した業務に応じて報酬が発生し得る。また，生田・前掲（注４）18 頁でも同旨の説明がなされている。

24） 裁判例ではシステムという用語が本稿におけるソフトウェアに相当する形で使用されているが，裁判例については原文そのままを引用している。

25） ここで「単体試験」「結合試験」は，前述の「単体テスト」「結合テスト」と同義である。

26） ユーザーの検収試験は，検収の可否の判断のために行われる試験であり，前出の「ユーザー受入テスト」に相当する。

27） 東京地裁平成 9 年 2 月 18 日判決（判タ 964 号 172 頁）は，瑕疵担保責任の事例において，「コンピューターソフトのプログラムには右のとおりバグが存在することがありうるものであるから，コンピューターシステムの構築後検収を終え，本稼働態勢となった後に，プログラムにいわゆるバグがあることが発見された場合においても，プログラム納入者が不具合発生の指摘を受けた後，遅滞なく補修を終え，又はユーザーと協議の上相当と認める代替措置を講じたときは，右のバグの存在をもってプログラムの欠陥（瑕疵）と評価することはできないものというべきである。」と判示し，ベンダーがバグに対応している限り，納品物に瑕疵は存在しないものと判断した。また，前掲注 23「ソフトウェア開発関係訴訟の手引」には，上記の判例を債務不履行責任にも及ぼし，たとえバグが発生していたとしても，ベンダーがこれに遅滞なく対応してバグを修補することが可能な場合には，ベンダーに債務不履行責任が発生しないというべきとの記載がある。

28） 情報システムの設計を，内部設計と外部設計に分ける場合がある。このとき内部設計は詳細設計を指し，外部設計は要件定義や基本設計を指す場合が多い。

29） 総合テストと同義である。

30） 他の参考判例として，東京地裁平成 22 年 5 月 21 日判決（判例集未登載，D1-Law 28181390），東京地裁平成 20 年 8 月 29 日判決（判例集未登載，2008 WLJPCA 08298023），広島地裁平成 11 年 10 月 27 日判決（判時 1699 号 101 頁）等がある。

31） 前掲（注 23）「ソフトウェア開発関係訴訟の手引」，佐々木茂美編『最新民事訴訟運営の実務』（新日本法規出版，2003 年）第 8 章参照。もっとも，システム開発には何千万円という多額の費用が掛かることから，ユーザーがリース契約を利用している場合が多い。そして，ユーザーはリース契約の関係上，ベンダーの開発が遅れても検収期間を遅らせることができず，きちんとテストができていなくとも検収印を押さざるを得ない場合があることから，常に検収をもってベンダーの仕事が完成したと判断することには疑問があり，ここでいう検収とは，実質的意味における検収，すなわち，ユーザーにおけるシステムのテストの合格という意味での検収を指すべきである。

32） 前掲東京地裁平成 16 年 3 月 10 日判決（判タ 1211 号 129 頁）は，「ベンダーは，システム開発の専門業者として，自らが有する高度の専門的知識と経験に基づき，本件電算システム開発契約の契約書及び本件電算システム提案書に従って，これらに記載されたシステムを構築し，段階的稼働の合意のとおりの納入期限までに，本件電算システムを完成させるべき債務を負っていたものである。」としており，ベンダーはシステム開発の専門家として契約の目的物を納期通りに開発すべき義務があるとされている。

33） ここでいう「内部統制」は，ベンダーにおけるソフトウェア開発過程の内部統制行為を指す。例えば，ソフトウェアの設計に関する各段階における設計内容の確認及び承認であったり，あるいはテスト計画の承認や承認されたテスト計画に従って実施したテスト結果の確認及び承認などの手続が該当する。

34） なお，全体の参考文献として，生田・前掲（注 4）の諸論文，前掲（注 23）「ソフトウェア開発関係訴訟の手引」，佐々木・前掲（注 31），及び以下の文献がある。

コンピュータ訴訟研究会『コンピュータ紛争事件のケース研究』（尚文社，1999 年），コンピュータ訴訟研究会『コンピュータ紛争事件のケース研究Ⅱ』（尚文社，2000 年），松村幸生「共同開発・委託研究におけるトラブル対策と契約書作成の実務（第 5 章）」『新版「ビジネス契約」実務大全』（企業研究会，2007 年），情報サービス産業協会法的問題委員会契約部会編『新しいソフトウェア

開発委託取引の契約と実務』（商事法務，2002 年），日本電子工業振興協会編『ソフトウェア開発モデル契約解説書』（コンピュータ・エージ社，1994 年），内布光「ソフトウェア開発委託契約紛争事例の研究（2）〈研究ノート〉」現代法学 11 号 93 頁（2006 年），同「ソフトウェア開発委託契約紛争事例の研究（1）〈研究ノート〉」現代法学 10 号 157 頁（2005 年）。