モジュールの知的財産権と
イノベーションのガバナンス

中　田　善　啓

甲南経営研究　第51巻　第２号　抜刷
平 成 22 年 11 月

モジュールの知的財産権とイノベーションのガバナンス

中　田　善　啓

１．は　じ　め　に

プラットフォーム・スポンサーが間接的なネットワーク効果を利用して，補完的製品の参入を促進するが[(1)]，スポンサーがデベロッパー（補完業者）の製品市場で競争するかもしれないというリスクがあるので，補完的製品とプラットフォーム・スポンサーの間に緊張関係がある[(2)]。したがって，プラットフォーム・スポンサーがデベロッパーを参入後に圧迫するかもしれないので，デベロッパーがエコシステムに参加するインセンティブに影響する。その理由は３つある。

１．デベロッパーはプラットフォーム・スポンサーに認証されないソフトウェアを生産するオプションがあるが，認証（certification）はプラットフォーム・スポンサーとそのソフトウェアを相互利用するというシグナルを提供する。エコシステムへ参加するデベロッパーのベネフィットは，アクセスからプラットフォーム・スポンサーのインストール・ベースを利用して販売できることである。

２．エコシステムのガバナンスはコーペティション（coopetition）の関係

（１）　Rochet and Tirole［2003］.
（２）　Gawer and Henderson［2007］.

にあり，競争と協調が共存している。プラットフォーム・スポンサーとデベロッパーは，類似した機能的モジュールを提供し，複数の製品市場で競争するかもしれない。

3．プラットフォーム・スポンサーによる補完的製品市場へ参入できるので，協調プロセスで知識の漏洩がおき，デベロッパーの知的財産権（(intellectual property, 以下 IP[(3)] という）が奪われるリスクがある。プラットフォーム・スポンサーのインストール・ベースへのアクセス，参入のリスク，知識専有の威嚇によってプラットフォーム・スポンサーの利益の増大は，エコシステムの協調行動のトレードオフとなる。

たとえば，メディア・プレーヤー・ソフトウェア市場では，WINAMP は Windows に基づく MP3 プレーヤーをリリースするために最初のソフトウェアであり，RealPlayer はインターネットについてのストリーミングメディアができる最初の Windows に基づくメディア・プレーヤーであった。しかし，両方とも無防備であったので，マイクロソフトは Windows Media Player をビルトインして，その製品市場に侵入して，利益を専有した。対照的に，主に川下の補完的能力をアップルが所有したために，iTunes（後にアップルがオープンにする）は，より大きい成功を得た。

一部の例外を除いて，モジュラリティ（modularity）に関する文献は，価値創造に焦点を合わせてきた[(4)]。対照的に，価値専有に関するモジュラリティの影響は，主に無視された。従来の研究ではプラットフォーム・スポンサーの観点から参加の問題を分析していた[(5)]。しかし，インテグラル・アーキテクチャからモジュラー・アーキテクチャ，同時にクローズド化からオープン化

（3） ここで，IP はモジュールの知的財産権を具体的に指している。
（4） 例外はここで主として取り上げる Henkel and Baldwin［2009］である。かれらは価値創造と価値専有との両面からモジュラリティをとりあげている。
（5） Gawer and Henderson［2007］.

ないしはハイブリッド化（一部をオープン化し，一部をクローズドにする選択的オープン化）が進行すると，プラットフォーム・スポンサーはどのように価値専有化をはかるか，また外部のデベロッパーをエコシステムに参加させて，集団的開発を行うかが重要になってくる。これは，プラットフォーム・スポンサーだけでなく，デベロッパーがエコシステムに参加することによって，価値を専有できるかが問題となる。

そこで，本論文ではオープン化の前提となるデベロッパーがエコシステムに参加する条件を考える(6)。価値専有化は焦点企業の戦略として参入障壁やイノベーションに関係する資産の所有（統合）に焦点が当てられてきたが，エコシステムは，むしろオープン化ないしはハイブリッド化は間接的なネットワーク効果を生み出すので，イノベーション・ガバナンスの転換を伴う。これらについては１節，２節，３節，４節で考察する。

モジュールの価値専有化を前提にすると，エコシステムがコミュニティとして開発を行う際に，プラットフォーム・スポンサーの役割を考える。これもデベロッパーのエコシステムへの参加が問題になる。プラットフォーム・スポンサーがエコシステムのメンバー（デベロッパー）と協調するようなアーキテクチャを構築することが問題となる。ここでは集団的開発，および開発の情報公開をゲーム論的状況で考える。これらについては５節で明らかにする。

２．エコシステムにおける IP の意義

2.1　モジュールと価値専有

多くの有形製品は，販売されると，買手はすべての使用権を得ることができるが，知的財産は無条件では移転されないことが多い。たとえば，機械の

（６）　実証研究については Huang, Ceccagnoli, Forman, and Wu［2009］を参照。

買手は可能な限りそれをフリーで使用するが，知識は具体化されていない。売手は，特許とライセンスしている規定によってこの知識を使用するために買手の権利を制限するかもしれないし，企業秘密にしてこの知識にアクセスを与えないかもしれない。企業は組織内でその IP を保持するかもしれないし，ライセンスするかもしれない。IP のライセンスは複雑で多様である。このような IP についてモジュラリティ（モジュール化）を行って，価値（利益）を専有することができる。

イノベーションへの投資を回収するには程度の差こそあれ，オープン化が必要になる。たとえば，1998年に，ゲームソフトウェア企業の Valve Software 社はゲーム「Half-Life」をリリースした(7)。そして，コアのエンジンと補完的なコードを別々のモジュールにするようなコード・ベースをデザインした。重要なことは，これらのモジュールは異なる条件下で，ライセンスされた。これは本論文で述べるが，モジュールが異なった IP ステータスを持っていることになる。

Valve 社はエンジンをライセンスして，機密のソースコードを保持したが，補完的ソースコードを公開して，ユーザーに，修正して，共有するためにユーザーに対し幅広いライセンスをデベロッパーに与えた。ユーザーは補完的コードを利用して，改良されたゲームは，Valve 社のオリジナルのゲームよりも普及した。ユーザーはフリーに修正できるのではなく，Valve 社のコアエンジンを利用しなければならなかったので，このような IP 志向的なモジュール化は成功した。インテグラルであれば，Valve はすべてのコード・ベースを公開するか，まったくしないかの選択しかない。前者であれば，利益が得られないし，後者では価値の共同創造をすることができない。

Valve 社はプラットフォーム・スポンサーとなって，IP 志向的モジュール

（7） 詳細は Henkel and Baldwin［2009］を参照。

化によって，デベロッパーと共同開発すると同時に，利益を獲得した。IP志向的モジュール化は多様であって，複雑な製品やプロセスによって，最適なIPステータスはシステムの部分によって異なる。最近の傾向としては，IPステータスのハイブリッド化がイノベーションを促進し，利益が拡大する。

イノベーションによる価値専有（利益獲得）はモジュラリティと知的所有権に関係する。通常特許，著作権，機密によって利益が専有される。イノベーションによってその投資を回収さなければならないので，オープン化は一般的に選択的に実行されるだろう。モジュラリティと価値専有との関係で，複雑なシステムの異なった要素（部品）に応じて，IPは多様に扱われる。

複雑なシステムはモジュールという要素に分解されるが，そのモジュール間の関係はモジュール内ではタイトなリンク，他のシステムの部分にルースなリンクをもつ。モジュールは，全体としてシステムが機能するように協調し，評価されなければならない。

複雑な製品ないしは，複雑なプロセスを実現するために必要な知識は，製品またはプロセスのモジュラー構造で上書き（overlay オーバレイ）される。具体的には，複雑なシステムでは，しばしば，当該IPと異なる知識の領域を扱うことが多い。このような戦略を混合IP（mixed-IP）戦略という。混合IP戦略は，各々互換的な知的財産だけを含むIPモジュールを創出すると同時に，IPモジュールと対応するように，基本的な製品またはプロセスのモジュールをデザインすることである。モジュール内ではIPのコンフリクトはない。

IPモジュラリティは，企業のすべてのレベルで重要である。焦点企業とその供給業者（顧客と補完企業）の間でコラボレーションの基本的な条件を設定するので，戦略的なレベルで重要である。IPモジュラリティは企業のビジネスモデルのベースとなる。製品デザインのレベルで，その人工物を設

計するとき，製品人工物はIPモジュラリティを考慮しなければならない。操作上のレベルで，組織はIPモジュラリティによって，混合IP戦略によって決定される選択的なオープン・イノベーション・プロセスに適応するために，そのルーティンを適応させなければならない。基本的に，エコシステムが価値の共同創造にとって効率的であるとき，およびコラボレーションする企業が広く分布して，特定することが困難なとき，IPモジュラリティがより有利である

イノベーションに関するIPマネジメントには2つの相対立する側面がある。特許と著作権の形のフォーマルなIPのマネジメントはイノベーションのマネジメントの重要な側面である。これとは逆に，オープン化，ハイブリッド化によるイノベーション・プロセスは，コントロール不能であることが多い。IPモジュラリティは，両方の傾向を調整するのに役立つ。

2.2　モジュラリティ

モジュラリティはタスクを分割する。タスクに分割することによって，分業は，企業の中の個人の間で，または，企業の境界間で発生する。製品とプロセスのデザインのモジュラリティは，複雑な製造工程でいくつかの企業間に分割する。明確に定義されたインターフェイスをもつタスク・モジュールが存在するので，取引費用は減少する。その結果，モジュラリティは，比較的低いコストで高い製品多様性を提供でき，ユーザーは構成要素をより優れたものと入れ替えることによって，選択的にその製品をアップグレードすることができる。

モジュールの範囲内の変化とモジュールの再結合は，モジュラー・アーキテクチャがあれば容易である。モジュラー・システムでは，イノベーターはモジュラー・アーキテクチャを創出した企業内のメンバーである必要はない。むしろ，エコシステムは，外部のデベロッパー，ユーザー，ベンチャーのイ

ノベーションを促進する間接的ネットワーク効果を利用する。したがって，関連企業だけでなく，ベンチャー，ユーザー，顧客がイノベーションを行うことができる。焦点企業から見ると，モジュラリティによって，イノベーションがその企業から，直接コントロールできないので，価値の創造と価値の専有の間に緊張関係が生まれることになる。

モジュラー・システムの人工物は，補完的イノベーターに価値（価格）の分配を失うリスクをもつ。たとえば，IBM のメインフレームコンピュータ System/360 のモジュラー・アーキテクチャによって，互換的な周辺機器（例えばディスク・ドライブ）のメーカーが参入することができた。IBM PC は非モジュラー・アーキテクチャもっていたが，大部分の価値を IBM ではなく，インテルとマイクロソフトが獲得した。しかし，以下で明らかにするように，イノベーターは，インターフェイスとコード間を分離するようなモジュラー・アーキテクチャを創出して，内部のコードをブラックボックス化して，インターフェイスを公開することによって，価値専有を容易にすることができる。

プラットフォームの分析ではプラットフォームの価値の創出，たとえばプラットフォームの利用価格などに焦点があてられてきたが，エコシステムは間接的ネットワーク効果を産み出すには，プラットフォーム・スポンサーを含め補完企業（モジュール）が，そのエコシステムに参加して，イノベーションを創出する誘因を考える必要がある。

2.3　障壁による価値専有

企業戦略について過去には大きな２つの流れがあった。１つは Rumelt（1984），Wernerfelt（1984），Barney（1991）などは企業の資源ベースを重視した。これは，競争優位性の源泉を，価値があって，稀少で，模倣不可能で，非代替的であるような資産，能力，組織プロセス，情報，知識に求める。利

益が参入障壁や資源ポジションの障壁から得られる。資源が知識であるとき，特許や著作権のように，知的財産権，機密が障壁となって，企業はバーゲニング・パワーを拡大する。このような資源ベース重視型は企業間で相互連結して，知識を共有するという考え方は取られなかった。それでも，企業間での知識共有の重要性が認識され，アライアンス・ネットワークのようなガバナンスが重視されている。

次に，Chesbrough and Teece［1996］, Langlois and Garzarelli［2008］は，価値の専有化の源泉として，イノベーションを重視する。イノベーションは，組織内の行われる自律的イノベーションと他組織内の部品などのイノベーションを必要とするシステマティックなそれに大別できる。換言すれば，自律的イノベーションはモジュラー・システムのモジュール内で発生し，システマティックなイノベーションはインテグラル・システムで発生する。

これら２つのイノベーションの性格は変わらない。本論文で取り上げるモジュラー・アーキテクチャはシステムの構造をインテグラルないしは自律的イノベーションに変化させることができる。

モジュラー・アーキテクチャには価値専有の問題がある場合がある。Teece［1986］によると，焦点企業が製品またはプロセスをモジュール化するならば，小企業が参入するのが可能で，モジュールでイノベーションが行われ，競争がおきる。オープン・アーキテクチャが取られると，この傾向はより強くなる。モジュラー・アーキテクチャはシステムのモジュール・レベルでイノベーションが行われ，それによって製品市場への参入が促進される。資源ベースの観点からすると，資源が分離され，資源ポジションの障壁が低くなって，競争が激化する。モジュラー・アーキテクチャは価値専有を重視しつつ，その本来のイノベーションを促進する方法をデザインする。そこでオープン・モジュラリティとクローズド・モジュラリティの２つのガバナンスが選択される。現実にはハイブリッドないしは選択的にオープンないしは

クローズドされていることが多い。

資源ベースのモデルと Teece［1986］のモデルはオープン化すると，イノベーションによる成果が拡散していくと考えて，障壁を設けることによって価値を専有するとされている。しかし，モジュール化は逆に，メンバーの間接的ネットワーク効果による利益を共有する。エコシステムのメンバーのイノベーションを共有する。当然，そこには，プラットフォーム・スポンサーによるデベロッパーの利益の専有，機会主義的行動，ただ乗りなどの負の外部効果が発生する可能性がある。以下ではこのような問題を考えていく。

3．IP モジュラリティ

3.1　IP ステータス

複雑なシステムはリンクされた要素からなり，それぞれは確実な IP ステータス（IP status）を実行する。IP ステータスは，モジュールについての知識と周囲のシステムでのその役割に対する法的権利とそれの事実上のアクセスを決定する。企業は他者に対して，どのようなタイプの権利を与え，どのようなタイプの知識にアクセスを強化するかを決定しなければならない。企業は，特許，著作権または秘密にして，その知識を所有権するかもしれない。さらには，第三者にその所有権のある IP をライセンスするか，組織内でのみ利用するかもしれない。また，オープンソース・ライセンスの範囲下で，IP を利用可能にするか，あるいは，パブリックに公開するかもしれない。

所有する知識を使用することに加えて，焦点企業は他者から関連した知識も得るかもしれない。そのような場合，要素のまたはモジュールの IP ステータスは，外部から与えられる。そのような IP はライセンスで獲得されるかもしれないか，オープン・ソースコード・ライセンスで得られるかもしれないし，パブリック領域からとられるかもしれない。

２つの要素の IP ステータスは，矛盾している場合がある。そのような IP

の非互換性に適応するために，それぞれの要素は，異なるモジュールが構築され，モジュールの境界が設定される。このようにして生成する IP モジュラリティによって，価値が専有される。

IP モジュラリティの概念は複雑な製品またはプロセスに適用する。そして，それはいくつかのコヒレント（coherent）な部品から成り，構成要素と呼ばれている。この構成要素はより小さな構成要素からなる。一般に，IP モジュラリティに基づく戦略はシステムの構成要素の間で強い補完性に依存する。製品またはプロセスは，全体として，その部品の合計より価値がある。

知識は，製品またはプロセスの基本的なモジュラー構造に対応して，分割される。さらに，知識が IP によってカバーされると，所有者には他者の使用を排除することができるので，関連した IP は基本的なモジュラー構造によって分割されることもある。

異なった知識を使用する IP は事実上の能力と矛盾している場合がある。組織に流入する IP については，役に立つ異なる条件を特定化し，組織から流出する IP については戦略の問題として，たとえば，IP の所有者がある部分のみを共有し，他の部分を共有するように，法的であるか契約のコンフリクトが発生する。このように，IP が矛盾しているとき，非互換的 IP という。たとえば，IP ステータスが特定の取引相手にのみ知られている場合，それは非互換的である。

すべての要素が互換性を持つ IP ステータスがあるならば，より大規模なシステムの特定のモジュールは IP モジュラーである。そのモジュールは IP モジュールである。システムのモジュールのすべてが IP モジュールであるならば，システムは全体として IP モジュラーであると言う。重要なことは，IP の非互換の情報が全体システムの異なるモジュールで関連する限り，この定義はシステムが IP がそれぞれ異なって，非互換的に使用されていても，IP モジュラーである。

IP モジュラリティは複雑な製品またはプロセスに適用され，コヒレントな要素（部品）がリンクするネットワーク・システムである。非互換的な IP が全体システムの異なったモジュールと関連している限り，システムがその異なった非互換的な IP を利用しているので，依然として IP モジュラーである。

図１(a)はインテグラルで，(b)が IP モジュラーである。図１のＡとＢは IP ス テータスであるが，非互換的である。

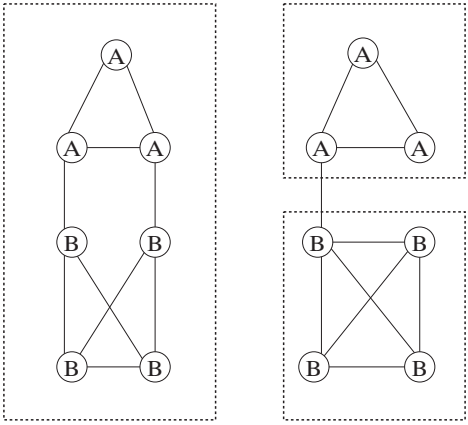

図１　インテグラルと IP モジュラー構造のシステム(1)

（注：要素Ａ，Ｂは特定の IP ステータスである。(a)はインテグラル構造であり，(b)は IP モジュラー構造である。点線は組織の境界を示し，(a)は１つの組織であり，(b)は境界をもつ２つの組織である。）

モジュラー IP のシステムはデザインないしは再デザインすることができる。システムの一部が IP モジュールでなくても，製品またはプロセスのモジュールである場合がある。IP モジュールは非互換的な要素を含む場合があるので，IP モジュラリティは，単純な製品またはプロセスのモジュラリ

ティより強い条件である。しかし，人工物はシステムのモジュール境界とインターフェイスをコントロールし，企業が問題の IP を所有するならば，異なった知識の集合に一致する IP ステータスをコントロールすることができる。

図１でインテグラル・システム(a)で，ＡとＢの依存関係が２つあるが，それを除去して１つにして，互換的な要素となるように，モジュールの境界を導入して，図１の(b)にすると， IP モジュラリティが成立する。図２で，オリジナル・システム(a)はモジュラー構造であるが，IP モジュラーでない。図２で，境界をシフトすることによって，図２(b)は IP モジュラー・システ

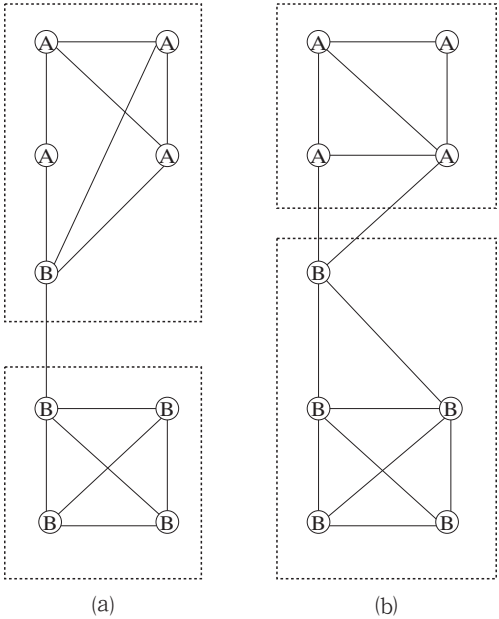

図２　インテグラルと IP モジュラー構造のシステム(2)

（注：要素Ａ，Ｂは特定の IP ステータスである。(a)はインテグラル構造であり，(b)は IP モジュラー構造である。点線は組織の境界を示し，両者とも２つの組織である。）

ムとなる。

3.2　IP モジュラリティの分類

IP モジュラリティは，表１に示すように，IP の所有と IP ステータスについての知識をもっているかどうかの２つの軸で分類される。IP の所有は，第１に，焦点企業から流出する IP と企業に流入する IP に分類される。システムにかかわる知識について，当該企業がその IP ステータスを既知か，未知のどうかである。流出する IP では当該企業は IP を所有し，各々の要素に有利な IP ステータスを与えるかどうか決定する。このように，他のエージェントに IP のバンドルと事実上のアクセスレベル，要素やエージェントに差別的に提供できる。

		IP ステータスの知識	
		既　知	未　知
IP の所有	流出 IP 要素の IP ステータスが特定化される	・M-System ・安定的コントロール・システム	Apple Darwin （OS）
	流入 IP 要素の IP ステータスが外部から所与	・ソフトウェア・プラットフォーム ・OSS としての JAVA	パテント・トロール

表１　IP モジュラリティのタイプ

当該企業に流入してくる IP ステータス場合にはこれと逆になる。IP の所有者は考慮する要素の IP ステータスを決定し，焦点企業は与えられた権利と利用可能性のバンドルを受け入れなければならない。

もし，製品のアーキテクチャが選ばれる時点で，焦点企業はモジュラー構造と IP コンフィギュレーションのどんな組合せがシステムの有効期間でもっとも有利かがわかっていれば，当該企業にとってその流出する IP は既知である。アーキテクチャが選ばれるとき，異なる構成要素の最適モジュラー

構造とIPステータスが既知か，シフトする需要または競争の威嚇によって変化すれば，そのIPステータスは未知である。望ましい流出するIPコンフィギュレーションが知られていないか不確実であるとき，システムのIPを指向するモジュール化は後日用いられることができるオプションを創出する。

表1でパテント・トロールは，ある企業が，その特許権を侵害している疑いのある企業（特に大企業）を見つけ出して，巨額の賠償金やラインセンスをとることである。Mシステムは次節で述べる。

4．IPモジュラリティ戦略

4.1　IPモジュラリティと価値の専有および価値の共同創造

2005年に，Israel-based M-System は，モバイルの携帯電話で広く使われている埋め込みのフラッシュメモリ製品に関してジレンマに陥っていた(8)。オープン・ソースコードOSのLinuxはモバイルのデバイスのために普及していって，フラッシュメモリの買手は，メモリーのドライバー（モバイルのデバイスの中で機能することができるソフトウェア）がオープンソース・ソフトウェアとして一般公開されていた。しかし，M-System はいわゆるフラッシュ・マネジメント・ソフトウェアに含まれるIPを保護したかった。そのため，製品の第1の世代のそれはドライバーでバンドル化された。

解決策として，M-System は，IPモジュラー・アーキテクチャを導入した。それはそのフラッシュメモリ・システムを再設計し，ドライバ・ソフトからフラッシュ・マネジメント・ソフトウェアを分離した。それらはフラッシュ・マネジメント・コードを物理的なメモリデバイス自体に配置し，著作権と機密によって保護しておいた。残りのドライバーは，オープンソース・ソフトウェアとして公開された。

（8）　詳細は Henkel and Baldwin［2009］を参照。

図１に示されるように，IP 非互換性(a)によって影響を及ぼされるインテグラル・システムは混合 IP 戦略(b)によって特徴づけられる IP モジュラー・システムに作り出された。そのフラッシュメモリ・システムの M-System の再設計は，いくつかの点でコストがかかった。

M-System にみられるように，価値を共同創造への可能性が存在するときはいつでも，価値のために共同創造を可能にする比較的オープン・モジュールへの製品と，価値取得を可能にする他の所有またはクローズド・モジュールに分割することは，意味がある。したがって，周囲のエコシステムの価値の共同創造向けの可能性が大きいほど，企業から流出する IP モジュラリティによって，オープン化とクローズド化が選択される。IP モジュラリティのベネフィットは，どの程度の範囲で価値が共同で創造されるかに依存する。要素が分散されていれば，オープン化が効率的である。

焦点製品がシステム・インテグレイターに販売されて，川下より大規模なシステムの部分になるとしよう。この川下のシステムが複雑であるとき，システム・インテグレイターは一般的に多数の既存の構成要素を必要とする。システム-レベル・パフォーマンスを最適化するために，システム・インテグレイターはしばしば個々の構成要素の行動を適応させなければならない。システム・インテグレイターが若干の詳細で構成要素を知っていて，理解するならば，そのような適応が可能になる。

しかし，構成要素のデザインのシステム・インテグレイターによる全面的な知識は，供給業者の IP を危険にするかもしれない。また，IP モジュール化は，このリスクを減らすことができる。構成要素を，その内部の働きだけに関係するインテグレイターに役立つ情報と，要素の内部の機能にのみ役だつ情報に分割することができる。IP 観点から，前者は比較的オープンな IP であり，後者が所有権のあるクローズドな IP である。

ユーザーが多様性を必要とするとき，すなわち，川下のユーザーのニーズ

が異質で予測できないとき，IP モジュラリティは重要な役割を果たす。多様性を求める顧客の要求に適応する方法は，顧客が選択するフィーチャまたはオプションのリストを焦点企業が提供することである。しかし，このアプローチはコストがかかって，あらゆる顧客ニーズまたは欲求をまだカバーしないかもしれない。

4.2　IP モジュールとプラットフォーム

異質で予測できないユーザーのニーズを扱うもう一つの方法は，製品と関連した流出する IP の 2 つのモジュールに分割することである。第 1 のモジュールは変更できず，その IP は保護されている。第 2 のモジュールはよりオープンな IP ステータス下で提供され，修正可能である。ユーザー・イノベーションのためのユーザーにツールキットを提供することのよって，ユーザーの修正は，それから，マイナーな変化からまったく新しいモジュールにわたるかもしれない[(9)]。

このような分離はプラットフォーム戦略に関連している。プラットフォーム・アーキテクチャにおいて，確実な構成要素（プラットフォーム）は，固定されているが，そのほかの要素（補完製品）が変化させることが可能である[(10)]。この戦略は，それから，ユーザーまたは第三者に補完製品のデザインにコントロールを与えることができる。プラットフォーム戦略は，所有者がプラットフォームのコアに所有権のある IP ステータスを与え，補完的構成要素をリンクさせるインターフェイスにオープンの IP ステータスを与えることが多い。

次にサプライチェーンの川上と共同で価値を創造するケースをみよう。川上で協調するためには，焦点企業は，そこから流出 IP（例えばその供給業

（9）　von Hippel［2001］.
（10）　Baldwin & Woodard［2009］.

者への青写真または部品仕様）を提供する必要がある。このとき，非公開契約やライセンスで知識は保護されるが，結局それは失敗する。IP モジュラリティは，知識と IP を分割することによって，焦点企業の一部の全体システムのための保護対策を再導入して，供給チェーンの価値を専有する能力を強化する。供給業者が焦点企業とその競争相手に販売するとき，IP の漏洩が起こるかもしれない。しかし，モジュールの境界は，供給業者が競争から顧客の製品を差異化するフィーチャに貢献している知識から，顧客のシステムで機能する構成要素を構築するのに必要な知識を分離する。

企業が外部の供給業者にその生産の大規模な部分を外注化するとき，供給業者にそのシステムのために製品に必要があるというデザイン知識を与えると，同時に，市場で競争する必要な知識を与えることになる。しかし，各々の供給業者は，構成要素に関連する知識にアクセスを必要とするだけである。このように，焦点企業が処理するならば，異なる供給業者は相互に情報交換しないようにするが，重要な IP の外注化は，各々の供給業者が一部の知識だけを知っていても，全体を再構築することができない。

IP モジュール化は，この場合異なるタスク・モジュールへの製品デザインまたは生産プロセスを分解し，異なる供給業者にそれらのタスク・モジュールを配分することを必要とする。このような分割統治に基づく IP モジュール化は，専有可能性が弱くなる可能性があっても，知的財産権の保護を強化することができる。モジュールの境界は別々のタスク・モジュールに製品を得るために必要なトータル知識を分割する。そして，それはそれから異なる供給業者に外注化されることができる。このような分割統治による IP モジュラリティは，国際的研究開発にも利用され，オープンと所有権モジュールを補完する。

IP モジュラー・デザインを創出することによって，企業も新しい価値の取得機会に応じてシステムの特定的部分でその IP 戦略を変化する能力を得

る。このように，IP モジュラリティは，オプション価値を創出する。通常，技術的または市場不確実性の高いレベルによって特徴づけられる設定では，外部の環境の変化は，望ましい混合 IP 戦略を事後的に得ることができる。たとえば，顧客は購入の条件として製品の IP の一部をオープン化の要求をするかもしれない，あるいは，生産のアウトソーシングはコスト理由のために魅力的になるかもしれない。初期のデザインでは，企業は混合 IP モジュラリティによって，環境の変化に対応できる。

あるソフトウェア企業が，全製品ファミリーが重要なプラットフォームの構成要素に依存しているとしよう。このプラットフォームは，もう一つのソフトウェア・ベンダーから会社の自身のコードならびにライセンス・コードを含んでいるので，この構成要素は IP モジュラーではない。さらにまた，コード・ベースを通してライセンス・コードを広めて，プラットフォームはインテグラルで設計される。このような状況では，ライセンスの更新と再交渉のとき，プラットフォームによるホールドアップ（holdup）のリスクがある。焦点企業は，この威嚇を予想して，コード・ベースを再モジュール化することができる。そして，別々のモジュールで新しくデザインされたプラットフォームとライセンス・コードを配置することができる。

システムが両方のモジュールを必要としたが，プラットフォーム・モジュールはもはや任意のライセンス・コードを含んでいない。その結果，焦点企業のスイッチング・コストは低下し，ホールドアップのリスクは小さくなった。このケースは資産特定性が高い。取引費用経済学では統合が効率的であるが，システムを IP モジュラリティはより効率的なシステムである。

当該組織へ多数で，多様な IP モジュールが流入し，相互に非互換的なケースがある。流入する IP の異なる情報と自分自身の IP の間の不適合性が増大するかもしれない。企業が入力する IP の多くの，多様なソースコードを持つとき，IP モジュラー方法でその全体のシステムを設計することによっ

て，コード・ベース，製品，プロセスであっても，IP 非互換性の影響を減らすことができる。モジュールの境界は，モジュールの範囲内で IP の適合性を保証して，再交渉のコストを最小にするために配置される。特に，多数の入力する IP のライセンスを再交渉しないで，モジュールの IP ステータスを変えることができる。

今日，企業は製品またはプロセス・アーキテクチャをますます選択するようになると，予測不可能な IP 関連の威嚇の影響を受けやすい。複雑な新製品またはプロセスのために，エレクトロニクスとソフトウェアのような分野では，確実に，製品が侵害する可能性のある特許をすべて確認するために不可能である。その結果，パテント・トロールがおきる。すなわち，イノベーターは予想外の企業から侵害申し立てを受け，損害賠償と法外なライセンス料を払う責任を負う場合がある。

基本的な製品またはプロセスがモジュラーであるとき，デザイン周辺はよりコストが低い。侵害が少数のモジュールに限定される限り，それらのモジュールは残りのシステムの機能を損なわないようにデザインすることができるか，除去することができる。対照的に，基本的な製品またはプロセスがインテグラルであるならば，全体のシステムの変化を引き起こす構成要素を再デザインすることになる。したがって，モジュラー製品またはプロセス・アーキテクチャではシステムの一部に対して，賠償金の請求のような予期しないホールドアップのコストを減らすことができる。不確実な IP ステータス下で要素が製品の別々のモジュールにカプセル化されるとき，それらがそのようにはるかに未確認の特許を侵害することが分かれば，それらを取り替えるか，とり下げるのが容易である。

5．非利他主義のゲーム

5.1　協調か非協調か

これまでの節ではエコシステムのIPモジュラリティについて述べてきた。次の問題は，エコシステムに参加して集団的開発がモジュールにとって，実際に利益が大きくなるかどうかである。

物理的な財と異なって，情報財特にソフトウェアは，１人のユーザーによるコードの利用が他のデベロッパーの利用を妨げないので，非競合的財である。プラットフォーム・スポンサー（プラットフォームの所有者），デベロッパー間の相互作用は，そのメンバーがソフトウェアの開発努力をするかどうかのゲーム論的状況を考えよう。このゲームは，プレーヤーが意図しないが，結果的に利他主義的行動をとった方が，自らの利益になるという特色をもっている。これは非自発的利他主義ゲームといわれる(11)。非自発的な利他主義者は，各々のプレーヤーの努力が他者の利益に貢献することであり，１人目のプレーヤーの行動が他者にプラスになるかどうかで，利益が生まれる。

対称的なデベロッパーがワンショットのゲームを始める。ペイオフについては不確実性は存在しない。２人のデベロッパーへのコードの価値はvである，そして，作成コストはcである。各々への価値は，コストより大きい（$v-c>0$）。したがって，他のデベロッパーがコードを作成しなくてもコードを書くためのインセンティブがある。さらに２つのケースを考える。１つは別個にコードを作成し，デベロッパーがコミュニケーションを行わず，コラボレートせず，独立して開発する。もう１つは２人のデベロッパーがゲームを行うケースである。

ゲームではデベロッパーがコミュニケートすることができ，各々のデベロ

（11）　Baldwin and Clark［2005］.

ッパーが他のコードを使用するのにコストがかからないと仮定する。さらに，２人のデベロッパーは，他者が書くコードを使用することを妨げることができない。コードが生み出されるならば，相手方に自動的に公開される。後者の仮定は，オープンソース開発プロセスにあてはまらない。オープンソース・デベロッパーは，それらが書くコードを公開することを強要されない。

まず，コードベース・アーキテクチャはモジュラーでなく，コードを書くタスクは分割できないと仮定することから始める。このゲームのペイオフマトリックスは表２で示されている。デベロッパーはコードを開発するか，開発しないかを選択する。後者はただ乗りである。このゲームには純粋な戦略で２つのナッシュ均衡があり，表２で（開発しない，開発する），（開発する，開発しない）である。この各々のデベロッパーが確率で機能するゲームの混合戦略均衡，α が存在する。開発するケースの期待値と開発しないケースの期待値を等しくして解くと，

$$\alpha^{*}=1-c/v.$$

デベロッパー２ ＼ デベロッパー１	開発しない	開発する
開発しない	0，0	$\boldsymbol{v}$, $\boldsymbol{v-c}$
開発する	$\boldsymbol{v-c}$, $\boldsymbol{v}$	$v-c$, $v-c$

表２　非自発的利他主義のゲーム

ゲームのプレーヤーの数が増大するにつれて，均衡は，予想できる方向で変わる。所与数のデベロッパー（N）について，N ヶの対称的純粋戦略均衡がある。そして，それは１つのデベロッパーが働き，すべてがただ乗りする特色を持つ。そして，１つの完全に混合戦略均衡が，常にある。N 人のデベロッパーの混合戦略均衡で，任意のデベロッパーが開発するという確率は，

$\alpha_N=1-(c/v)^{1/N-1}$。N が増大するにつれて，この数は減少する，しかし，デベロッパー当たりの期待値は常に $v-c$ である。

表３は，デベロッパーがコミュニケーションをしないで，孤立してコードを書くケース，純粋戦略でフリーライダーのデベロッパーとコードを書くデベロッパーの利益，混合戦略の利益が示されている。したがって，これは非自発的な利他主義者ゲームの側面を示している。

独立開発	$v-c$
純粋戦略	
フリーライダー	v
開発者	$v-c$
混合戦略	$v-c$

表３　開発のペイオフ

また，混合戦略の期待値は独立してコードを書くケースと同じ利益であるので，期待レベルではデベロッパーは集団的努力に参加することで利益を得られない。現実に，期待値レベルでの協調的行動から利益が得られないのは問題である。協調してコード作成の作業をする前に，他のデベロッパーより高い利益を得るか，低い利益を得るかの考えるデベロッパーは参加する。ところが，事後的に，すべてコストを負担し，利益が得られないデベロッパーが集団にとどまるか，退出するか無差別になる。

5.2　プラットフォームの役割

モジュラー・アーキテクチャがゲームを変更する方法を見るために，１つのプラットフォームと，ＡとＢの２つのモジュールからなるコード・ベースを考えよう。ゲームの時に，プラットフォームが存在するので，そのコストはサンクする。定義上，各々のモジュールは，別々に機能する。共同開発されるならば，各々は価値の半分を得て，コストの半分を負担する。このよう

に，プラットフォームと２つのモジュールから成っているネットワークは，上述の非モジュラー・システムと同じ価値と全体的なコスト（プラットフォームを排除する）を持つ。

しかし，プラットフォームさらにどちらのモジュールでも，機能的システムである。価値をシステムで実現するためにすべてのモジュールが必要ではない。まず，第１に各々のデベロッパーは，一度に１つのモジュールで機能すると仮定する。第２に，利用可能になるとすぐに，すべてのデベロッパーはそのコードを公開すると仮定する。

各々のデベロッパーは，以下を選ぶ

⑴　開発しない

⑵　Ａのタスクを開発する

⑶　Ｂのタスクを開発する

完全情報の下でのペイオフマトリックスは表４のようになる。純粋戦略のナッシュ均衡は，表４で（Ａの開発，Ｂの開発）と（Ｂの開発，Ａの開発）である。このタスクはデベロッパー１と２に公平に配分され，フリーライダーは存在しない。しかも，デベロッパーは異なってコードを作成する。さらに，デベロッパーが別々にコードを作成するときのペイオフが，$v-c$ となるが，ネットワークに参加すると，$v-0.5c$ となるので，両デベロッパーはネットワークに参加する。

デベロッパー２ ＼ デベロッパー１	開発しない	Ａの開発	Ｂの開発
開発しない	0, 0	$0.5v$, $0.5(v-c)$	$0.5v$, $0.5(v-c)$
Ａの開発	$0.5(v-c)$, $0.5v$	$0.5(v-c)$, $0.5(v-c)$	$\boldsymbol{v-0.5c}$, $\boldsymbol{v-0.5c}$
Ｂの開発	$0.5(v-c)$, $0.5v$	$\boldsymbol{v-0.5c}$, $\boldsymbol{v-0.5c}$	$0.5(v-c)$, $0.5(v-c)$

表４　プラットフォームをもつ非自発的利他主義ゲーム

次に，大規模なシステムを考えよう。j をコードベース・アーキテクチャのモジュールの数であり，N は潜在的デベロッパーの数である。デザイン・ルールとプラットフォームがすでに存在すると仮定しよう。問題はすべてのモジュールがコード化するかどうかである。

完全情報で，純粋戦略のゲームではすべてのモジュールは１回だけコード化する。あるモジュールがコード化されない戦略のプロフィールを考えよう。すべての他のプレイヤーの戦略を固定すると，モジュールが $(v-c)/j>0$ であるので，コード化すれば，あるデベロッパーの効用は高くなる。したがって，コード化されない戦略は均衡ではない。いくつかのモジュールが１回以上コード化する戦略を考えよう。そのモジュールをコード化するデベロッパーの１人はモジュールをコード化しないで，効用が上がる。したがって，１回以上コード化する戦略プロフィールは均衡ではない。

次に，モジュール内の作業に難易度があるタスクの配分をプラットフォーム・スポンサーが行うかどうか，すなわちそのような戦略プロフィールがナッシュ均衡かどうかである。

モジュールがコード化する配分は，$(N_j)!/(N_j-j)$ ある。すべてのプレーヤーがいったんその戦略にコミットする同時手番ゲームにおいて，これらの配分の各々は，デベロッパーがそのペイオフを一方的に改善することができないナッシュ均衡である。しかし，これらのナッシュ均衡は，自己強制的な戦略か，時間整合的戦略に依存する。これらはいわゆるサブゲーム・パーフェクト均衡である。そこにおいて，時間内の各々のポイントの戦略は対応するサブゲームの均衡を構成する。サブゲーム・パーフェクト均衡は，プレーヤーが実行するのが難しい約束に依存しない。この理由で，多期間設定では，サブゲームパーフェクト均衡は，自分に有害である威嚇または約束を実行しているデベロッパーに依存する均衡よりも，現実的な場合がある。

プラットホームが公平な開発のタスク集合と不公平な開発タスクの集合に

分割するとしよう。不公平な均衡に，開発負担の大きいデベロッパーは，最後の期間まで待機して，１つのモジュールをコード化することによって改善する。最後の期間には，残りのモジュールがコード化する。結果的に，若干のモジュールがコード化されないままの限り，最後の期間で開発しないという他のデベロッパーによる威嚇は，最後の期間のサブゲームで均衡と整合しない。すべての高い作業負担のデベロッパーは，待機後にコード戦略を行う方がベターオフである。このように，初期の不公平な配分は，サブゲーム・パーフェクトでない。

要約すると，同程度の割引率で N 人のデベロッパーを伴っている純粋な戦略の j 期間ゲームにおいて，すべてのサブゲームパーフェクト均衡では，デベロッパーの間でタスクの公平な配分が行われる。モジュラー・コード・アーテクチャ（$j>1$）があれば，これは，仕事ができるだけ公平に分割されるだろうことを意味する。$N>j$（モジュールより多くのデベロッパー）ならば，j デベロッパーは開発するだろう。そして，フリーライダーの数は縮小するだろう。$N\leq j$（デベロッパーより多くのモジュール）ならば，すべてのデベロッパーは開発するだろう，そして，フリーライダーがない。

ここではフリーライダーが集団的開発プロセスの部分である労働者のインセンティブに重要でない。参加して，集団にとどまって，コード化する要因は，第１に，たとえば同程度の割引率をもつデベロッパーが存在することである。第２に，このような集団的開発システムは異なるモジュールで全体として機能するように，プラットフォーム・スポンサーが設定するモジュラー・アーキテクチャをもつ。

5.3　非自発的利他主義のオプション価値

ソフトウェア開発は，デザイン・プロセスである。デザインの基本的特性は，デザイン・プロセスの初めは，最終的な結果が不確実である。不確実性

は，新しいデザインがオプションの性格をもつ。オプションは行動方針を選んで，関連の報酬を得る権利である。新しいデザインは，新しい方向で何かするための必要条件――義務でなく権利――ではなく，適応能力を生み出す。しかし，新しいデザインは，採用されるとは限らない。定義上，モジュラー・アーキテクチャはモジュール・デザインが変わっても，全体としてシステムの機能を下げないで改善される。この意味で，モジュラー・アーキテクチャは不確実性に対応できる。

コード・ベースのモジュラー構造とオプションは観察可能であり，かつ実験可能である。事後のモジュラー構造は，いろいろなツールを用いたコードベースの依存関係をマップすることによって見つけることができる。オプションに関して，新しい機能が加えられる（たとえば Linux の新しいデバイス・ドライバー），あるいは，既存の機能がアップグレードされるたびに，デザイン・オプションは用いられる。

コードベースのアーキテクチャがデザイン規則に関してフォーマルに指定されたならば，そのモジュラー構造を決定することは簡単である。オプションはモジュールに依存する。モジュラー構造が既知であれば，オプションを確認し，コードベースに固有のオプション価値を推定できる。

他方，多くのオープンソース・プロジェクトは，そのアーキテクチャのデザインルールを正式に特定化しない。それにもかかわらず，デベロッパーはドキュメント化されている n ヶのコードベースのモジュラー構造を認めることができ，オプションのロケーションと大きさについて情報に基づいた判断をすることができる。その評価は，初期の人工物にコミットしているかもしれない。たとえば，コードベースには，階層アーキテクチャがあるかもしれない。また，Linux が Unix に基づいたので，新しいコードベースは有名なアーキテクチャのモデルに基づく。

最後に，デベロッパーはコード・ベースの上で直接機能することによって，

モジュラー構造とオプション・バリューについて学習することができる。変更によってコードが少なくなれば，コードベース・アーキテクチャはモジュラーであり，他の状況が等しければ，オプション・バリューは高くなるだろう。変化がシステムを通して分岐して，全体の深い知識を必要とする傾向があるならば，コードベース・アーキテクチャはモノリシック（monolithic）である。[12] 他の状況が等しければ，オプション・バリューは低いだろう。オープンソース・コードベースの間で，GnuMeric の表計算プログラムが比較的モノリシックであるが，Apache ウェブサーバはモジュラーである。

これまではシステムとモジュールの価値に不確実性が存在しないと想定した。完成デザインの価値には不確実性がある。以下ではプレーヤーの行動と結果として生じる均衡に関して不確実性とオプション価値の影響を調べる。デベロッパーはリスク中立的で，期待値最大化であり，我々がその時間選好を無視することができるように，コード化している期間が十分に短いと仮定する。

まず，モジュラリティがなくても，オプション価値が十分にある場合，デベロッパーは参加して，集団的開発努力にとどまることに対して，正のインセンティブがある。その理由は，デザイン結果が不確実であるとき，パラレル作業が必ずしも重複しても，冗長であるとは限らない。独立したデベロッパーは，別のデベロッパーが開発した優れたコードを利用することができない。したがって，プロセスに参加して，集団に参加してインセンティブをも

（12）　モノリシックカーネルは OS 全体が一体化している。これに対し，カーネルには最小限の機能しか持たせず，なるべく多くの機能を外部モジュールとして提供する設計手法をマイクロカーネルという。機能がモジュールとして独立しているマイクロカーネルより開発効率は劣るが，プロセス間通信などのオーバーヘッドが少ないため，実行速度では勝るとされる。ただし，最近では，モノリシックカーネルの OS が機能をモジュール化する手法を取り入れたり，モノリシックカーネルにひけを取らない処理効率を実現したマイクロカーネルなどが登場している。

つ。

オプション価値が高いモジュールの組み合わせはモジュラー・システムの点で固有の能力で組み合わせ，開発するあらゆるデベロッパーのインセンティブを増大する。デベロッパーが固定的にプールされていれば，これらのインセンティブは，ただ乗りするデベロッパーは減少する。このように，高いオプション価値によるモジュラー・コード・アーキテクチャはフリーライダーを抑制する。そのようなアーキテクチャも，別のコーディングに対して，集団的プロセスの中で開発するあらゆるデベロッパーの優位性を増大する。

コード・ベースの価値が基本的な構成要素の実現された価値の合計となり，構成要素価値分布は，モジュールの離散的構成要素の組合せとなるアーキテクチャ下で同じである。構成要素結果は，完全に相関していないような条件の下では，構成要素確率分布の任意の集合について，モジュラー・アーキテクチャは，対応するモノリシックのアーキテクチャより高い期待値を持つ。(13)

次に，モジュールとオプション価値が増えるにつれて，参加するデベロッパーは増大する。モジュラー・システムがベストのデベロッパーの組み合わせが行われると，デベロッパーは集団開発へのインセンティブが大きくなる。このように，高いオプション価値によるモジュラー・コード・アーキテクチャがフリーライダーを抑制して，集団での開発が行われ，独力で開発するよりも優位である。

所有権のある企業と集団的開発プロセスの製品は，競争するかもしれない。顧客ベースが大きい所有権のある企業は，より多くのデザイン探索を実行して，集団的開発プロセスより高品質のシステムを提供するだろう。しかし，集団的開発プロセスの価値はシステムの使用から得られるわけではない。価値は所有権のある企業と，集団開発システムとの競争から生まれる。基本的

（13） 証明については Baldwin and Clark［2005］を 参照。

なアーキテクチャがモジュラーでかつまた高いオプション価値を持つとき，価格は高い。

同じようなアーキテクチャをもつ集団的開発と，所有権のある企業との競争を考えよう。均衡で企業は大きな顧客ベースをもつので，集団的開発よりデベロッパー数は多くなる。研究開発が行われた後では開発コストがサンクしているので，ユーザーのそのコストはフリーである。そのような競争に直面すると，所有権のある企業は，その価格を下げる。価格切り下げによる利益は開発者とフリーライダーが獲得する。

利己的なデベロッパーが集団的開発努力に参加し，その効果が所有権のある企業への価格引き下げのプレッシャになる。基本的なアーキテクチャが非常にモジュラーでかつまた高いオプション価値を持つとき，それらの集団的努力ではデベロッパーは大きい価値を得る。このように，それらのプロジェクトがコスト見合う価値を提供するので，デベロッパーは高いオプション価値でモジュラー・アーキテクチャの集団的開発プロジェクトに誘引されるであろう。

5.4　コードの公開

ここまで，デベロッパーがコード・ベースの任意のモジュールで開発するならば，そのコードがコード化している期間の終わりに他のデベロッパー（フリーライダーを含む）に自動的に公開されると仮定した。この仮定下で，システムがモジュラーであるか，十分なオプション価値をコード化しているモジュールのコストに比較して，協調開発努力が継続されることを明らかにした。

デベロッパーは開発したコードを公開する必要はない。さらに，コミュニケーションのコストが，常にある。コミュニケーション・コストは小さいかもしれないが，必要である。さらにまた，他者にとって利用できるためには，

貢献されたコードは重要なコード・ベースに統合されなければならない。統合のこのコストの部分は，しばしばコードの開発者が負担しなければならない。

2つのモジュールが成果を公開公表するかどうかをゲームで考えよう。全システムのデベロッパー価値を v とすると，1つのモジュールの価値 $0.5v$ である。コードを公開する1人のデベロッパーのコストを $0.5r$ とする。コードをコミュニケートし，システムにモジュールを統合するコストは，r に含まれる。このゲームのペイオフマトリックスは表5に示される。このゲームはワンショットの囚人のジレンマ・ゲームである。ナッシュ均衡は（公開しない，公開しない）である。

デベロッパー2 / デベロッパー1	公開しない	公開する
公開しない	**$0.5v$, $0.5v$**	v, $0.5(v-r)$
公開する	$0.5(v-r)$, v	$v-0.5r$, $v-0.5r$

表5　非自発的公開ゲーム

このような囚人のジレンマ・ゲームではプラットフォーム・スポンサーがデベロッパーにベネフィットを与えて，公開しない戦略のペイオフを小さくすることができる。たとえば，プラットフォームが $f>r$ となるようなベネフィット f を与えれば，均衡は（公開，公開）となる。f は貨幣的利益だけでなく，多様な形をとり，利他主義，他者関連的な選好関数，規範（norm）に基づくネットワークのガバナンスが存在する[(14)]。評判，発明発見の喜びなどである。Harhoff, Henkel, and von Hippel［2003］によれば，オープンソース開発プロセスが私的財（販売のために生産される）とは異なり，オープンソ

（14）　Fehr 等［1999, 2000, 2002, 2007］, 中田［2004］, Fauchart and Hippel［2006］を参照。

ース・プロセスは，デベロッパーがもっとも必要なコードを開発して，コード化するかを選ぶという点で，それ自体が報酬となる公共財を生産する。

6．お　わ　り　に

オープンかクローズドかの議論には本論文で明らかにしたIPモジュラリティの議論をまずはじめなければならない。モジュールのIPステータスがオープンか，ハイブリッドか，クローズドかを決定する。モジュールがあるからといって，クローズドでもないしオープンでもない。

モジュールは部品単位だけではなく，知識，情報の単位であるので，IPが問題となる。モジュールが物理的であれば，取引で所有関係は移転する。ところが，それが物理的な部品や製品に加えて，情報や知識が一体となっていたり，モジュールが情報や知識であれば，利用やアクセスについて制約がある。その明確な形式がIPステータスである。

日本のベンチャが育たない理由の１つはIPステータスが確立されていないので，大企業ないしはプラットフォーム・スポンサーがデベロッパーの利益を専有していることが多いことである。さらには，日本の独占禁止法が大企業寄りなので，そのような利益の専有が多くなる。

IPモジュラリティの概念は知識や組織に適用できる。後者の典型は組織のファイアウォールである。知識のIPモジュラリティは中小企業のエコシステムか，大企業と中小企業とのエコシステムの指針となる。

以上のようなIPモジュラリティの議論を前提にすると，プラットフォーム・スポンサーの役割が重要になる。５節で述べたように，それはエコシステムでどのように協調関係を形成するかである。これらは非自発的利他主義が働くので，集団的開発が機能する。問題は，その開発の成果をエコシステムのメンバーに公開するかどうかである。これは囚人のジレンマ・ゲームとなるので，プラットフォーム・スポンサーの役割が重要になる。

エコシステムで完全なオープンは少なく，ハイブリッド化が多い。しかし，参加するメンバーがイノベーションの成果を自社で囲い込み，それが参入障壁となって，利益を専有するパラダイムから，エコシステムのメンバーがイノベーションを共有するパラダイムへしつつある。

参 考 文 献

Baldwin, C. and K. Clark, “The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?” Harvard Business School Working Paper, (“The Architecture of Participation: Does Code Architecture Mitigate Free Riding in Open Source Development Model?” *Management Science,* 52, pp. 1116-1127.)

Baldwin, C. and Woodard, J. [2009], “The Architecture of Platforms: A Unified View,” in A, Gawer (ed.), *Platforms, Markets and Innovation,* Edward Elgar, pp. 19-44.

Barney, B. [1991], “Firm Resources and Sustained Competitive Advantage,” *Journal of Management,* 17, pp. 99-120.

Chesbrough, W. and D. Teece [1996], “When Is Virtuous? Organizing for Innovation,” *Harvard Business Review,* 74, pp. 65-73.

Fauchart, E. and E. von Hippel [2006], “Norm-based Intellectual Property Systems: the Case of French Chefs,” MIT Sloan Schools Working Paper 4576.

Fehr, E. and K. Schmidt [1999], “A Theory of Fairness, Competition and Cooperation,” *Quarterly Journal of Economics,* 114, pp. 817-868.

Fehr, E. and S. Gächter [2000], “Fairness and Retaliation: The Economics of Reciprocity,” *Journal of Economic Perspectives,* 14, pp. 159-181.

Fehr, E. and H. Gintis [2007], “Human Motivation and Social Cooperation: Experimental and Analytical Foundation,” *Annual Review of Sociology,* pp. 43-64

Fehr, E. and U. Fischbacher [2002], “Why Social Prefernces Matter－The Impact of Non-Selfish Motives on Competition, Cooperation and Incentives,” *Economic Journal,* 112, pp. 1-33.

Gawer, A. and Cusumano, M. A. [2002], *Platform Leadership, How Intel, Microsoft, and Cisco Drive Industry Innovation,* Havard Business School Press（小林俊男訳『プラットフォーム　リーダーシップ』有斐閣　2005年）.

Harhoff, D., J. Henke, and E. von Hippel [2003], “Profiting from Voluntary Information Spillovers: How Users Benefit by free Revealing Their Innovation,” *Research Policy,* 32, pp. 1753-1769.

Henkel, J. and C. Baldwin [2009], “Modularity for Value Appropriation: Drawing the Boundaries of Intellectual Property,” Harvard Business School Working Paper, 09-097.

Huang, P., M. Ceccagnoli, C. Forman, and D. J. Wu [2009], “Participation in a Platform

Ecosystem: Appropriability, Competition, and Access to the Installed Base," NET Institute, at http://ssrn.com/abstract=1480900.
Langlois, R. and Garzarelli [2008]," Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration," Working Paper.
中田善啓［2009］,『ビジネスモデルのイノベーション：プラットフォーム戦略の展開』同文舘.
中田善啓［2004］,『市場的ネットワークと互恵性』『甲南経営研究』44，pp. 1-34.
中田善啓［2009］,『ビジネスモデルのイノベーション――プラットフォーム戦略の展開』同文舘.
中田善啓［2010］,『イノベーションのガバナンス』『甲南大学経営学部50周年記念論集』（未刊）
Rochet, J-C. and J. Tirole [2003], "Platform Competition in Two-Sided Market," *Journal of the European Economic Association,* 1, pp. 990-1029.
Rumelt, R. P. [1984], "Toward a Strategy Theory of the Firm," in B. Lamb (ed.), *Competitive Strategic Management,* Prentice-Hall.
Teece, D. [1986], "Profiting from Technological Innovation: Implications for Integration, Collaboration Product Development," *Research Policy* 15, pp. 285-305
von Hippel, E. [2001], "Perspective : User Toolkits for Innovation," *Journal of Product Innovation Management,* 18, pp. 247-257.
Wernerfelt, B. [1984], "A Resource-Based View of the Firm," *Strategic Management Jouranl,* 5, pp. 171-180.