App Storeでの公開に向けた開発

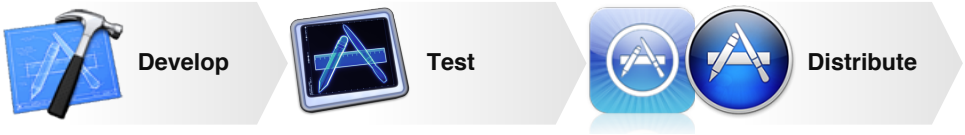
Developer

目次

図、表

アプリケーションの開発プロセスについて

iOS用、Mac OS X用のアプリケーションは、かつてないほど開発しやすくなりました。Appleが提供する、開発作業支援ツール群が充実しているからです。このドキュメントでは、開発チームの編成から、実際に利用したユーザの声に応えるところまで、アプリケーション開発の各段階を概観します。



At a Glance

このドキュメントでは、App Storeで公開するアプリケーションの開発プロセスをひと巡りします。読み終わったときには、開発チームを編成してアプリケーションを設計、実装し、App Storeに公開するまでの手順が頭に入っていることでしょう。

Appleプラットフォーム向けの開発では、管理作業と実装作業が車の両輪を成す

アプリケーション開発期間の大部分は実装（コードの記述）作業に費やすことになりますが、その間にもさまざまな管理作業をこなす必要があります。1人の開発者が両方行うというやり方もありますが、開発チーム内で分担してもよいでしょう。

管理作業の多くは、開発プロジェクトの立ち上げ時と終結直前に集中しています。たとえば、初めてのアプリケーション開発であれば、ある1人がAppleとの契約に署名して、Appleデベロッパ（認定開発者）になる必要があります。この人を**チームエージェント**と呼びます。チームのために法律的な責任を負い、参加者を集め、それぞれの責任と権限を定めます。

関連する章 「App Storeに公開するアプリケーションの開発」 （9 ページ）

App Storeで公開するためにはAppleの認定を受ける必要がある

App Storeでは販売するアプリケーションを制限しています。良質のアプリケーションだけをユーザに提供できるよう配慮しているのです。簡単にクラッシュしたり、大きなバグが露見したりするようなアプリケーションは販売できません。公開手続きの大部分は、アプリケーションを登録し、Appleの認定を受ける作業です。

関連する章 「アプリケーションをApp Storeで公開」 （42 ページ）

App Storeで公開するアプリケーションにはデジタル署名が必要である

コード署名を施すのは、ユーザ、開発チーム、Appleのそれぞれにとって、セキュリティ保全上の利点があるからです。署名を施したアプリケーションは、不正な改変を防止できます。攻撃者が何らかの改変を施すと、コード署名が無効になるため、実行できません。また、悪意あるコードが混入していた場合に、責任の所在を明らかにする効果もあります。iOS用、Mac OS X用ともに、App Storeで公開するためにはコード署名が必要ですが、iOSではさらに厳格になっています。開発期間中であっても、署名がなければiOSデバイス上で実行できません。

開発チームを編成する際、チームエージェント（またはその委任を受けた者）は、アプリケーションに署名を施す権限を持つメンバーを指名し、必要なリソースを作成する必要があります。

関連する章 「開発チームの立ち上げ」 （11 ページ）、「プロジェクトの作成」 （23 ページ）

実装作業の多くはXcode上で行う

Xcodeには、コードの記述、デバッグ、ユーザインターフェイス設計の機能が、ひとつの開発環境に統合されています。したがって、開発期間を通してXcodeを使うことになります。登録して認定を求める段階でも、実際の作業はやはりXcode上で行います。Xcodeをインストールすると、付属するほかのアプリケーションも使えるようになり、品質向上に役立てることができます。たとえばInstrumentsというアプリケーションには、アプリケーションの動作を記録、分析するためのさまざまなツールが組み込まれています。アプリケーションが想定通りに、効率よく動作していることを、組織立てて確認できるので有用でしょう。

関連する章 「プロジェクトの作成」（23 ページ）、「アプリケーションの開発」（31 ページ）

管理作業はさまざまなリソースを利用して行う

チームの管理に当たっては、さまざまなリソースを利用します。その主なものを以下に示します。

- 「Member Center」ウェブサイトは、主にチームエージェントが、開発チームにメンバーを加え、各々の権限レベルを設定するために使います。
- 「iOS Provisioning Portal」（Mac OS X向けの開発の場合は「Developer Certificate Utility」アプリケーション）は、コード署名に必要なリソースの作成に利用します。チーム管理者（チームエージェント自身、またはその作業を委任された者）は、このツールを使ってコード署名リソースを作成し、メンバーに提供します。
- 「iTunes Connect」ウェブサイトは、アプリケーション開発のうち、ビジネス寄りの情報を管理するために使います。販売/財務情報、App Storeでの販売時に表示される情報、Appleのサーバに格納される情報などです。チームエージェントは、「Member Center」と同じように、各々が「iTunes Connect」にアクセスする権限を設定します。

関連する章 「開発チームの立ち上げ」（11 ページ）、「プロジェクトの作成」（23 ページ）、「アプリケーションをApp Storeで公開」（42 ページ）

アプリケーションの動作の多くは、コードではなくデータによって定義される

アプリケーションがApp Storeにどのように表示され、実行したときどのように動作するかを決めるのは、コードだけではありません。データも大きく寄与します。さらに、非動作時のアプリケーションをオペレーティングシステムがどのように表示するかも、データによって決まるのです。

データのいくつかは、実行形式ファイルとは別のファイルに格納されます。このように、データとファイルを一体化して扱えるようにしたものを、**アプリケーションバンドル**と言います。Appleのサーバに格納されるデータもあります。たとえば、App Storeにアプリケーションを表示するための情報は、主としてiTunes Connectに格納されています。データの格納場所はさまざまですが、重要なのは、アプリケーションが単に実行形式ファイルだけから成るのではない、ということです。むしろ、コード、データ、サービスが一体となったもの、と考えるべきでしょう。

関連する章 「プロジェクトの作成」（23 ページ）、「アプリケーションの開発」（31 ページ）、「アプリケーションをApp Storeで公開」（42 ページ）

How to Use This Document

開発チームにおける役割にかかわらず、全員がこのドキュメントを読んで、App Storeで公開するアプリケーションの開発手順を把握しておいてください。

プログラマは、事前にいずれかのアプリケーションチュートリアルを読んでおけば、このドキュメントに書かれている考え方をよく理解できるでしょう。

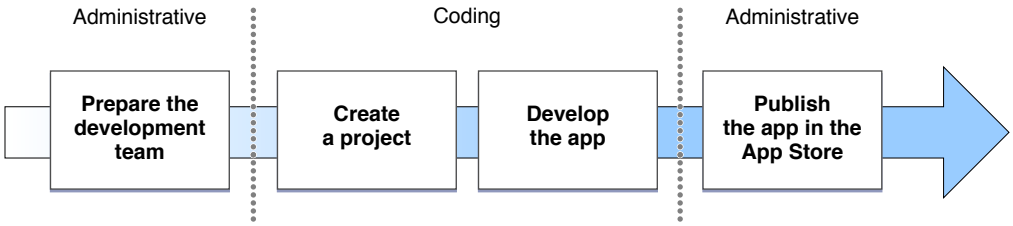
- iOS： *Your First iOS App*
- Mac OS X： *Your First Mac App*

App Storeに公開するアプリケーションの開発

以下、アプリケーションを開発し、App Storeに公開する手順を示します。ここでは論理的な順序で手順を示しますが、アプリケーションや開発チームの都合で、順序を入れ替え、さらには後戻りしなければならない場合もあります。たとえば途中でチームに誰かが加われば、手順を戻して若干の管理作業を行うことになるでしょう。

図 1-1に大まかな開発プロセスを示します。各ステップは大まかに、管理ステップと実装ステップに分類できます。管理ステップは通常、チームエージェントまたはその委任を受けた者が実施します。実装ステップは主としてプログラマが実施します。

図 1-1　開発プロセスの概要



以下、各ステップについて簡単に説明します。その後、それぞれ章を改めて、詳しい作業内容を説明します。

開発チームの立ち上げ

ある1人をチームエージェントとして指名し、新しい開発チームを編成します。チームエージェントは次に、ほかの者をチームに加え、それぞれのアクセス権限を設定します。最後に、開発者がアプリケーションに署名を施すために必要な、コード署名証明書その他のリソースを用意します。

プロジェクトの設定

続いて、アプリケーション開発の場である**Xcodeプロジェクト**を作成します。アプリケーションの種類に応じて適切なテンプレートを選択し、初期設定をカスタマイズします。また、アプリケーションに署名を施し、公開するために必要なリソースも作成します。

アプリケーションの設計や、組み込む予定のサービスによっては、ほかにも何らかのアプリケーション設定を、この段階で済ませておく必要があるかも知れません。というのも、Appleが提供する技術の中には、コードを記述するだけでなく、ある種のデータも用意しないと意図通りに使えないものがあるのです。たとえばApple Push Notification Serviceを利用するためには、特別な証明書を作成してサーバ上に置き、利用に当たって認証を受けられるようにしなければなりません。このサービスを使うことがあらかじめ分かっているならば、この段階で証明書を作成しておくとよいでしょう。

アプリケーションの開発

アプリケーション開発と聞いて、多くの人がまず思い浮かべるであろう作業に当たります。ユーザインターフェイスや機能を設計し、コードの形で実装し、想定通りに動作するかどうかテストします。プログラミング作業に多くの時間を費やすことになりますが、ほかにも、アプリケーションの動作に必要なデータアセットを作成するなど、重要な作業があるでしょう。具体的には、アイコン、音声ファイル、アートワークなどのアセットがあります。アプリケーション自身が直接使うものだけでなく、オペレーティングシステムが必要とするものもあります。また、ほかの言語やロケールに対応してローカライズしなければならないアセットもあります。設計段階では、どのようなアセットが必要か、ローカライズが必要なのはどれか、を把握することも重要です。

順次改良を施して、公開可能なアプリケーションに仕上げます。

アプリケーションをApp Storeで公開

アプリケーションをApp Storeに公開するためには、若干の管理作業が必要です。まず、当該アプリケーションを、App Store上にどのように表示するか決めなければなりません。また、アプリケーションをAppleに登録し、認定を受ける必要があります。認定が得られたら、公開（販売）の開始日を設定します。その後はAppleのツールを使って、販売状況、ユーザからの要望、クラッシュレポートなどを随時参照し、対応してください。バグを修復し、次の版でどのような改善を施すか検討するために役立つでしょう。バグ修復版や大きな改訂版の用意ができたら、初回と同様に認定を受け、公開日を設定します。

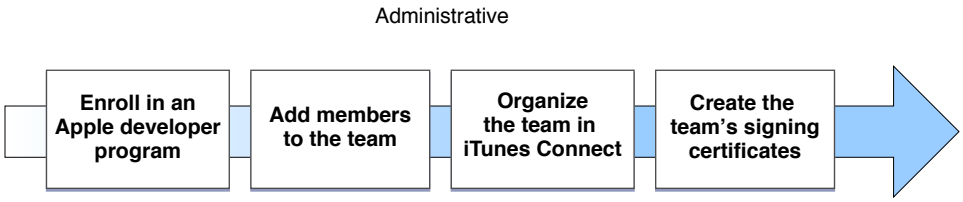
開発チームの立ち上げ

「開発チーム」は一般に、プログラマ、設計者、意匠画家、作曲家など、さまざまな人が集まってアプリケーションを開発しているグループのこと、と考えられています。iOSやMac OS Xのアプリケーション開発チームもそれで間違いではないのですが、これですべてではありません。「Apple Developer」ウェブサイト上で作成、管理しているグループ、という意味合いもあるのです。開発の各段階では、この意味の「開発チーム」を対象とする作業も随時発生します。チームのメンバーが1人だけであれば、こういった作業はすべて自分でしなければなりません。大人数のチームであれば、作業や責任を分担してもよいでしょう。

開発チームの一人ひとりに、「Appleデベロッパプログラム」の認証に用いる、一意的な**Apple ID**を割り当てます。また、アクセス権限も個別に設定できます。たとえば、App Storeに公開するなどの重要な作業を、特定の人しか実施できないよう制限することが可能です。

この章では、チームの各員に割り振るさまざまな役割を説明します。その後、どのように編成を進めればよいか、ガイダンスを示します。最後に、チームに属する所定の人が、アプリケーションに署名を施せるようにするための作業を説明します。図 2-1に具体的な手順を示します。

図 2-1　開発チームを作成する手順



Appleデベロッパプログラムへの登録

まず、ある1人が、iOSまたはMac OS Xのデベロッパプログラムに登録します。この人が**チームエージェント**になります。両方のオペレーティングシステム上で動作するアプリケーションを開発するのであれば、2つのプログラムに登録しても構いません。登録の過程でチームエージェントは、Appleデベロッパになるための法的な契約書に署名し、App Storeからの売上金決済に必要な財務書類を作成します。

チームエージェントは特別な立場で、「開発チーム」を対象とするあらゆる作業ができる一方、法的な責任も負います。また、チーム編成作業の大部分も行います。チームにほかの人を加えた後、必要であればこの権限を委任し、作業を代行させることも可能です。

Important チームエージェントは法律的にチームを代表する人なので、「Apple Developer」ウェブサイト上で降格させたり、権限を制限したりすることはできません。チームエージェントを交代させたい場合は、Appleに直接連絡してください。

アプリケーションに特別な技術を組み込む場合など、ライセンス契約を更新したり、新たに締結したりして、チームエージェントの署名を求められることがあります。たとえばiAdサービスを利用する場合、別途契約書に署名しなければなりません。

チームにほかの人員を追加

チームエージェントは、デベロッパプログラムに登録した後、ほかの人員をチームに加え、それぞれの権限を設定します。チームに属するのがチームエージェント1人だけであれば、追加の設定は必要ありません。チームエージェント自身があらゆる機能を利用できるからです。しかし、以降の開発プロセスで必要になることがあるので、どのような作業があるか、ひと通り読んでおくとよいでしょう。

ある人をチームに加える際、チームエージェントは招待状を送るという手続きを取ります。その過程で、チームにおける権限も設定します。招待状を受け取った人は、その時点で自動的にチームに加わります。

チーム各員の権限について

チーム各員の「権限レベル」とは、Appleデベロッパプログラムのウェブページや、そこに格納されたチーム情報に対する、アクセス権限のことです。この権限レベルによって、チームを代表して実行できる作業の種類が決まります。たとえば、アプリケーションをApp Storeに公開する権限を、特定のメンバーにのみ与える、といった管理が可能です。このように、ある作業を行う権限を制御できるようにしているのは、チームのセキュリティを管理しやすくするのが目的です。

チームが複数のデベロッパプログラムに加わっている場合は、それぞれ別々に権限レベルを設定してください。同じ人が、あるプログラムについては権限を持ち、別のプログラムについては持たない、という設定も可能です。

表 2-1に、チームの各員に割り当てることができる「役割」と、その簡単な説明を示します。いずれも、それより下位の役割に割り当てられた権限は行使可能です。

表 2-1 チームの役割

役割	説明
チームエージェント	**チームエージェント**は法律的にチームを代表し、Appleとの第一義的な連絡窓口になります。ほかの人員の権限レベルを変更できます。
チーム管理者	**チーム管理者**はほかの人員の権限レベルを設定できます。ただしチームエージェントの権限を奪うことはできません。アプリケーションに署名するためのアセットを管理します。開発段階か、配布できる状態になった段階かを問いません。開発用デバイス以外でも動作するような形で署名できるのは、チーム管理者に限ります。また、チームメンバーからの署名証明書要求を認定する権限もあります。
チームメンバー	**チームメンバー**は、Appleがプログラムのポータルサイトで配布する、一般には未公開の情報にアクセスできます。また、開発中のアプリケーションに署名できます。ただし、開発用署名証明書を要求し、チーム管理者が認定した後に限ります。

表 2-2に、チームの各員に対して与えられる権限を詳しく列挙します。このドキュメントでは、すべての権限は説明していませんが、最後まで読み通せばおおむね趣旨は分かるでしょう。

表 2-2 役割に応じた権限

権限	チームエージェント	チーム管理者	チームメンバー
法律的にチームを代表	✓	✗	✗
Appleとの第一義的な連絡窓口	✓	✗	✗
チーム管理者、チームメンバーを招待	✓	✓	✗
開発用署名証明書の要求を認定	✓	✓	✗
開発やユーザテストに用いるデバイスを追加	✓	✓	✗
アプリケーションIDを作成	✓	✓	✗
配布用署名証明書をAppleに要求	✓	✓	✗
開発用/配布用プロビジョニングプロファイルを作成	✓	✓	✗

権限	チームエージェント	チーム管理者	チームメンバー
アプリケーションIDを有効化（Apple Push NotificationやIn-App Purchaseに使用）	✓	✓	✗
SSL証明書を作成（Apple Push Notification Serviceに使用）	✓	✓	✗
開発用署名証明書を要求	✓	✓	✓
開発用プロビジョニングプロファイルをダウンロード	✓	✓	✓
一般未公開の情報を閲覧	✓	✓	✓

iTunes Connectにおけるチームの編成

前節で示した各権限は、主として開発プロセスに関するものでした。アプリケーションに署名できる人、署名証明書を作成できる人などを規定しています。一方、チームエージェントは、「iTunes Connect」ウェブサイトのアクセス権限も管理します。たとえば、アプリケーションの販売価格を変更する権限は、ごく少人数に制限するべきかも知れません。そこで、「iTunes Connect」ウェブサイトのアクセス権限は、「Member Center」とは独立に、しかもよりきめ細かく設定できるようになっています。ここでもやはり、チームの各員にいくつかの役割を割り当てることができ、役割ごとにそれぞれ異なる権限が対応しているのです。表 2-3に、それぞれの役割を大まかに説明します。

表 2-3 iTunes Connectにおける役割と責任

役割	責任
Legal（法務）	チームエージェントに自動的に割り当てられる役割です。ほかの人には割り当てることはできません。法的な契約書その他に署名する権限です。
Admin（管理）	「iTunes Connect」におけるあらゆる作業（法的代表に割り当てられているものを除く）ができます。チームエージェントには必ず割り当てられる役割であり、チームエージェントを交代させない限り、この役割を奪うことはできません。「iTunes Connect」における役割を、各員に割り当てることができます。
Finance（財務）	財務報告書や販売情報にアクセスできます。また、契約、税務、銀行取引などの情報を閲覧できます。
Sales（売上）	販売データにアクセスできます。

役割	責任
Technical（技術）	「iTunes Connect」に格納されたアプリケーション情報を編集し、In-App Purchaseを扱う機能のテストに必要な、テストアカウントを作成できます。

表2-4に、「iTunes Connect」を通して頻繁にアクセスする必要が生じるモジュールと、それにアクセスできる役割を示します。役割として「法的代表」が載っていないのは、この役割を担うのがチームエージェントだけだからです。「iTunes Connect」のアカウントに格納された自分自身の個人情報は、誰でも編集できます。

表 2-4 iTunes Connectのモジュールとそれにアクセスできる役割の関係（概要）

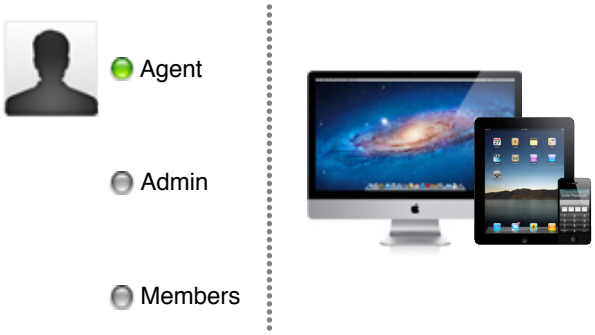
責任	Admin（管理）	Finance（財務）	Sales（売上）	Technical（技術）
Manage Users（ユーザの管理）	✓	✗	✗	✗
Manage Your Applications（アプリケーションの管理）	✓	✗	✗	✓
Manage Test Users（テストユーザの管理）	✓	✗	✗	✓
Sales and Trends（売上や動向）	✓	✓	✓	✗
Contracts, Tax, and Banking（契約、税金、銀行）	✓	✓	✗	✗
Payments and Financial Reports（決済、財務報告）	✓	✓	✗	✗

開発チームの編成

「Member Center」および「iTunes Connect」における各員の役割が分かったので、次に、どのようにチームを編成するかを考えましょう。チームの編成は、上述のような権限の設定だけで終わるわけではありません。規模によって、チームやアセットの編成方法は異なります。

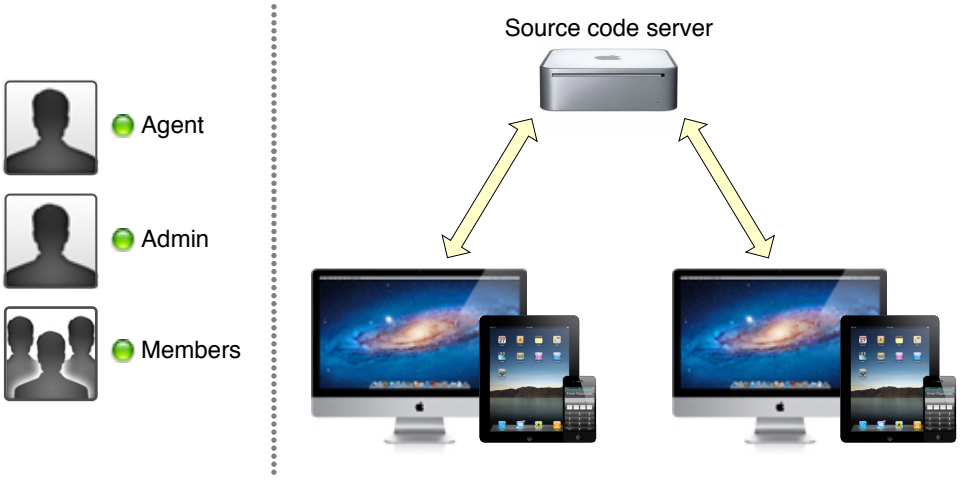
最も小規模なのは、1人だけのチームです。その1人がチームエージェントとなり、あらゆる権限を持っています。逆に言うと、作業はすべて1人で行わなければなりません。アプリケーションの署名や公開に必要なアセットを用意し、「iTunes Connect」に必要な情報を設定し、アプリケーションを開発し、販売しなければならないのです。

図 2-2 1人だけのチーム：その1人がチームエージェントになる



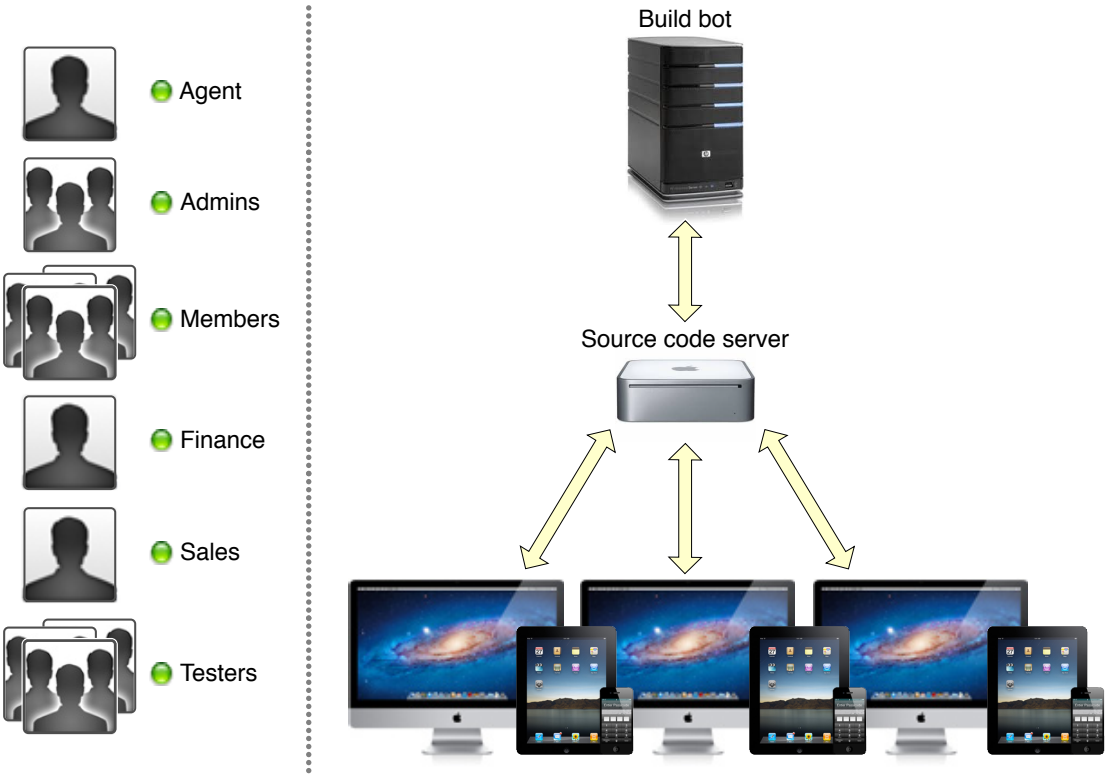
より一般的なのは、小人数の開発チームでしょう。このようなチームでは、チームエージェントがプログラマを兼ねますが、管理作業の一部をほかの人に委任します。財務や販売の作業はチームエージェントが行う一方、チーム管理者が、開発に必要なアセットその他を確保します。

図 2-3 小規模な開発チーム



チームが大規模になれば、さらに作業を分担できます。チーム管理者を含め、プログラマでない人も加わるでしょう。販売や財務の役割は、そういった業務の経験がある人に割り当てた方がよいかも知れません。チーム内にテスト担当者（非プログラマ）を参加させることも考えられます。アプリケーションの開発や公開は、専任のチーム管理者に委ねても構いません。さらに、iOS用とMac OS X用に人員を分けることも検討してください。

図 2-4　大規模な開発チーム



チームが大きくなると、人員間の調整が必要になります。特に、プログラマが増えるにつれ、開発中に障害が発生した場合の損失も大きくなります。たとえば1人だけのチームであれば、1台のコンピュータにファイルをすべて格納し、その中で作業を完結させてもよいでしょう。Xcodeには、ローカルにソースコードのリポジトリを作成し、コードを保存する機能もあります。しかし人数が増えるとどうなるでしょうか。

ある程度以上の規模になると、何らかの開発基盤が必要になります。ソースコードのリポジトリを置くリモートコンピュータを、別に用意する必要があるでしょう。開発者はそこから自分のコンピュータにコードを取り込み（チェックアウト）、変更後、サーバに戻す（チェックイン）ことになります。しかし、何人かが同時に同じコードを編集すると、チェックインの際に衝突が起こる可能性があります。問題が起こりにくいよう、開発ブランチをいくつか用意して管理するのが普通です。安定版ブランチには出荷用のコードを置いておき、実験版や開発版のブランチ上で開発作業を進めるのです。そのためには、ソースコードのリポジトリについてより詳しく把握し、ブランチ間でコードを統合する際の方針を決めておかなければなりません。

チームがさらに大規模になると、誰かが問題のあるコードを開発版ブランチにチェックインしたとき、チーム全体の生産性が損なわれる恐れがあります。問題が顕在化した頃にはすでに、ほかの開発者のコンピュータに伝搬しているかも知れません。エラーを修復したとしても、チーム全員に反映されるまでには多少時間がかかります。このような事故の頻度と、そのために失われる時間は、チームの規模が増すにつれて深刻になります。これが積み重なると、チーム全体の開発効率にも影響を与えかねません。これを避けるためにも、コード管理のための基盤機能がさらに必要となるでしょう。たとえば継続的インテグレーション（継続的統合）という技法があります。ビルド専用のコンピュータ（ビルドボット）を用意し、変更があるたびに自動的にチェックアウトして、アプリケーションをビルドし直す、というものです。これにより問題点を早期に発見できます。問題があるコードは、誰かがチェックアウトする前に、元に戻すようサーバを設定するとよいかも知れません。ユニットテストや自動ユーザテストの仕組みを採用していれば、ビルド専用コンピュータで随時実行し、ソースコードに問題がないことを検証することも考えられます。このようなプロセスを*スモークテスト*と呼び、開発ブランチに格納されているコードが正常にビルド、実行できる、という意味の信頼性を確保するために役立ちます。

チームの規模が大きくなるほど、頻繁に発生し、結果が事前に予測でき、手作業ではコストがかかる処理を自動化すれば、効果は絶大です。チームの負担が減り、設計や実装に集中できるようになるのです。

チームの署名証明書を作成

App Storeで配布するアプリケーションには、暗号技術に基づくデジタル署名が必要です。開発中に署名が必要になることもあります。

- Mac OS X用アプリケーションでは、たとえばiCloudやアプリケーションサンドボックスを使っている場合がこれに当たります。
- iOSの場合はあらゆるアプリケーションで、開発中でも署名が必須です。

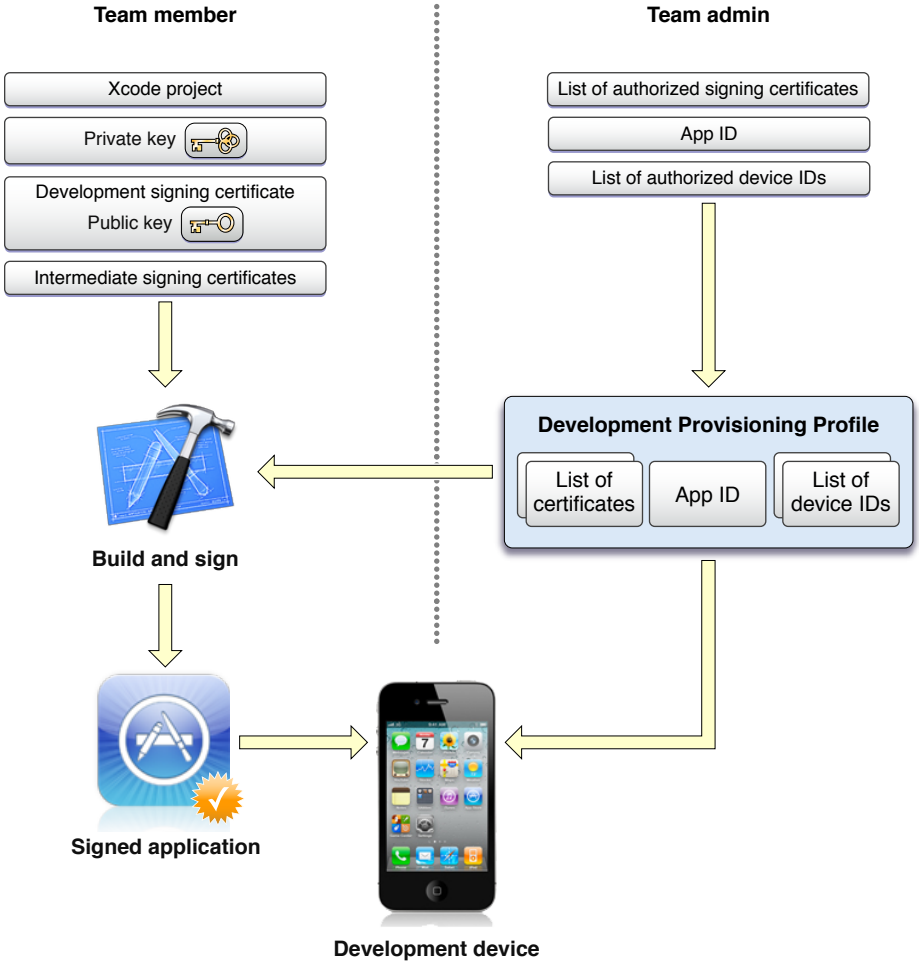
いずれの場合も、チームの全員がコード署名の手順を理解していなければなりません。この節では、アプリケーションに署名し、デバイス上で実行できるようにするために必要なアセットを挙げ、チームの各員がこれを入手する方法を説明します。アセットの中には、チーム編成後すぐに作成するべきものと、プロジェクトごとに作成するものがあります。

例：開発中のiOS用アプリケーションの署名

図2-5に、iOS用アプリケーションの開発において、ビルドや署名のために必要な作業やアセットを示します。Mac OS X用アプリケーションの場合も、また、配布するアプリケーションについても、同様の手順で署名しなければなりません。ここではiOS用アプリケーションの開発を念頭に説明します。開発中のiOS用アプリケーションは、開発用である旨の印がついたiOSデバイスにしかインストールで

きません。実際にアプリケーションを動かすためには、署名済みのアプリケーションと、その起動時に認証を行う**開発用プロビジョニングプロファイル**を、当該デバイスにインストールする必要があります。プロファイルがなければ、iOSは起動を許可しません。

図 2-5 iOS用アプリケーションの開発プロビジョニングプロセスの概要



開発用プロビジョニングプロファイルの作り方は「プロジェクトの作成」（23 ページ）で説明します。ここでは、これが署名を施したファイルで、次の事項が定義されている（認証できる）ことを理解してください。

- 実行可能なアプリケーション。
- 当該アプリケーションが実行可能なデバイスのリスト。
- アプリケーションの署名に用いた開発用証明書のリスト。

プロファイルは開発用デバイスだけでなく、開発用コンピュータにもインストールしなければなりません。これは、当該コンピュータ上のXcodeが、アプリケーションの署名を認証するために使います。Xcodeがアプリケーションに署名し、デバイスにインストールするのは、署名証明書および署名対象のアプリケーションを、プロファイルが認証できる場合に限ります。これは合理的な振る舞いと言えるでしょう。アプリケーションの起動を認証するプロファイルがなければ、署名済みアプリケーションをインストールしても、何の役にも立ちません。

次に、図 2-5の左側に着目してください。開発中、コードに署名するためには、開発システムのキーチェーンに、以下に示すものを組み込む必要があります。

- 秘密鍵。
- **開発用署名証明書**（チーム管理者が認証済み）。この中に、上記の秘密鍵と組になる公開鍵が含まれています。
- 中間署名証明書（Appleが提供）。この証明書は、開発用署名証明書と、署名証明機関との仲介をします。

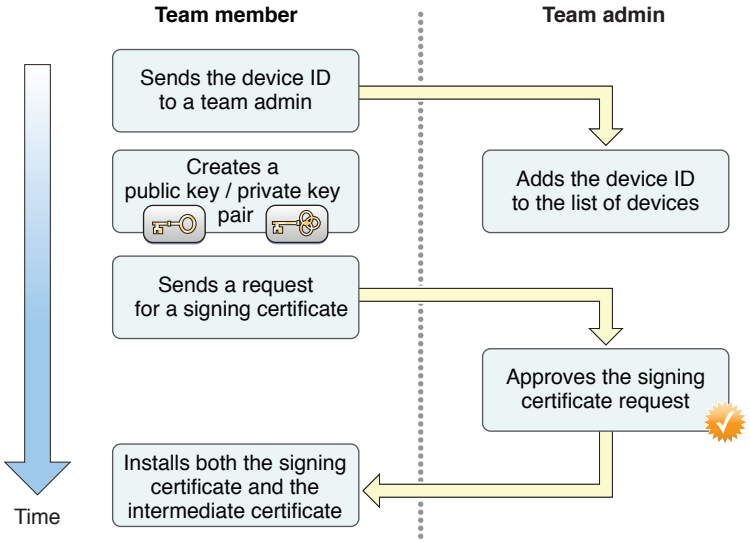
署名証明書と対応する秘密鍵の組を、**コード署名ID**と言います。iOS用アプリケーションをビルドする際、Xcodeはこれをコンパイルした後、コード署名IDを使って署名し、デバイスにインストールします。プロビジョニングプロファイル（署名済みアプリケーションと適合するもの）もこのデバイスにインストール済みであれば、アプリケーションを起動し、デバッグできるようになります。

コード署名を行う開発者の設定

図2-6に、チームの新しい開発者が作業を進められるよう、必要な設定をする手順の概要を示します。詳細は「より詳しい資料」（22 ページ）に示すドキュメントにまとめてあります。ここでは大まかな手順のみ把握しておいてください。チームメンバーが要求を出し、チーム管理者が認証または却下

する、という流れになります。認証の手続きは大きく2つに分かれます。あるデバイスをチームの「開発用」として追加する手続きと、開発中のアプリケーションに署名するために必要な、コード署名IDを生成する手続きです。

図 2-6　デバイスの認定を要求し、署名用証明書を取得する手順



あるデバイスを「開発用」として追加するためには、当該デバイスの**一意デバイスID（UDID、Unique Device ID）**をチーム管理者に伝えなければなりません。この作業は、Appleが提供するツールの管理範囲外です。したがって、たとえば、電子メールで送信する、という方法でも構いません。チーム管理者は当該デバイスを、チームのデバイスとして追加します。通常、チーム管理者はこのデバイスを、チームが管理する開発用プロビジョニングプロファイルにも追加します。最もこれは、必須ではありません。チームによっては、アプリケーションが動作するデバイスを制限することもあるからです。この場合、チームのプロファイルではなく、そのサブセットに追加するだけです。

チームメンバーは、署名証明書が必要であれば、Xcodeを使って要求することになります。Xcodeは自動的に公開鍵と秘密鍵を生成し、チームメンバーのために開発用署名証明書を作成するよう要求します。証明書要求を受け取ったチーム管理者がこれを承認すれば、Xcodeを使って自動的に、証明書をキーチェーンにインストールできるようになります。

ここではセットアップを一連の手続きとして説明しましたが、チームの全員が両方を実行しなければならないわけではありません。たとえば、「Apple Developer」ウェブサイト上ではチーム内テスト担当者を開発チームに加えていても、アプリケーションのビルドや署名はさせない、という方針も考えられます。この場合チーム管理者は、テスト担当者のデバイスを開発用として追加するけれども、コード署名IDは生成させないことになります。外部のテスト担当者も同様に、そのデバイス上でアプリケーションを実行できるようにするためには、当該デバイスをチームのデバイスとして追加しなければなりません。

以上は開発用署名証明書に関する説明でしたが、チームの編成時には、**配布用署名証明書**も要求する必要があります。これが必要なのは、開発用以外のデバイス向けにアプリケーションを配布する場合です。開発用署名証明書の場合はチームメンバーごとにコード署名IDが割り当てられますが、配布用アプリケーションの署名には、チーム全体で共通のコード署名IDを使います。配布用署名証明書の作成手順も上記と同様ですが、要求を出すのはチーム管理者（開発用とは別の鍵ペアを使用）、承認するのはAppleである、という点が違います。Appleが承認した後、証明書をダウンロードできるのはチーム管理者だけです。

Important　チームの各開発者は、秘密鍵をそれぞれ、安全な場所にバックアップしておかなければなりません。なくしてしまうと、新たにIDを生成しない限り、コードに署名することはできなくなります。さらに、鍵が盗まれてしまうと、何者かが開発者を騙って署名する恐れがあります。チームの配布用署名証明書に対応する秘密鍵の保護は*非常に重要*です。最悪の場合、何者かがチームを装い、悪意あるソフトウェアをアップロードして認定を得ようとするかも知れません。秘密鍵はキーチェーンにのみ格納し、紛失してもほか人に悪用されないようにしてください。

チームの開発者が別の開発用コンピュータでコードに署名する場合は、当該コンピュータに秘密鍵をコピーしなければなりません。

より詳しい資料

iOS用アプリケーションの開発チームに属している場合は、チームの編成や署名証明書の設定に関して、以下の資料を参照してください。

- チーム管理者：『*iOS Team Administration Guide*』。
- チームメンバー：『*Tools Workflow Guide for iOS*』。
- iTunes Connectおよびここで実行する作業の包括的な解説：『iTunes Connect Development Guide』。

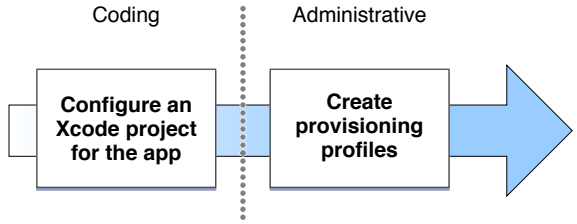
Mac OS X用アプリケーションの開発チームに属している場合は、チームの編成や署名証明書の設定に関して、以下の資料を参照してください。

- チーム管理者およびチームメンバー：『*Tools Workflow Guide for Mac*』。
- iTunes Connectおよびここで実行する作業の包括的な解説：『iTunes Connect Development Guide』。

プロジェクトの作成

開発工程を学習し、その考え方を体験するためにアプリケーションを構築する場合は、以下の正式な手順に従わなくても構いません。アプリケーションをApp Storeで公開する予定であれば、事前にいくつか作業を済ませておく必要があります。図 3-1に、新規アプリケーションの開発を始めるに当たって実行するべき手順を示します。この章では、プロジェクトを編成するための手順、アプリケーションを配布するために必要な関連作業について説明します。次に、各手順に関係する重要な考え方を説明します。

図 3-1 新規プロジェクトの開始



開発するアプリケーションに合わせてXcodeプロジェクトを設定

Xcodeプロジェクトを新たに作成した後、さまざまな設定ができます。設定可能な事項について詳しくは、『*Tools Workflow Guide for iOS*』および『*Tools Workflow Guide for Mac*』を参照してください。アプリケーション開発に用いるテンプレートの選択や、これに関連するプロジェクト設定は、慎重に行う必要があります。最初に適切なテンプレートを選べば、開発作業が順調に進むはずです。たとえば、OpenGL ESグラフィックスフレームワークを使ってiOS用のゲームを開発する場合、「OpenGL ES Application」テンプレートを選ぶとよいでしょう。OpenGL ESアプリケーションには必須の初期化コードが大量に組み込まれているからです。

新たに作成したプロジェクトには、アプリケーションのバンドルIDも設定しなければなりません。すると自動的に、プロジェクトの情報プロパティリストファイルに、必要な記述が追加されます。

注意 新規プロジェクトを作成すると、ローカルなソースコードリポジトリを自動生成するか否かの選択肢が現れます。ほとんどの場合、テストプロジェクトであっても、これはオンにするべきでしょう。ソースコード管理は、開発を効率的に進める上で不可欠です。開発作業の履歴が確実に残るので、必要があればいつでも変更を元に戻すことができる、という前提でコードの書き換えを進めることが可能です。また、ソースコードの作業コピーを、リポジトリに保存されているものと比較するのも容易です。

アプリケーションのプロビジョニングプロファイルを作成

アプリケーションをApp Store上で公開する予定であれば、チーム管理者はプロジェクト作成後、次の作業をする必要があります。

1. 特定アプリケーションIDを作成する。これは、バンドルID照合文字列の部分が、開発するアプリケーションのバンドルIDと完全一致するアプリケーションIDのことです。
2. このアプリケーションIDを使って、開発用プロビジョニングプロファイルを作成する。
3. 同様に、配布用プロビジョニングプロファイル（必要ならば複数）を作成する。

特定アプリケーションIDは、必須ではありません。しかし、対応するプロビジョニングプロファイルと併用すれば、各アプリケーションを誰がビルドし実行できるのか、きめ細かく管理できます。また、Appleサービスを利用する際に必要となることもよくあります。早い段階で作成しておけば、こういったサービスをいつでも組み込めることになります。

配布用プロビジョニングプロファイルは、実際に配布する際に作成しても構いません。しかしこれも同様に、プロジェクトの開始時点で作成しておけば、署名やプロビジョニングが適切にセットアップされていることを、早い段階で確認できます。また、アドホック配布用プロファイルをプロジェクト開始時に作成しておけば、いつでもリリース版をビルドし、開発用以外のデバイスでテストする、外部のテスト担当者に送ることができます。

アプリケーションに関する重要な用語

プロジェクトの作成に当たって、個々のアプリケーションを識別する**バンドルID**と、これに一致する**アプリケーションID**を使います。アプリケーションIDは、プロビジョニングプロファイルに記載して、当該アプリケーションを起動してよいとの認証を得るために必要です。以下の各節では、この3つの用語について詳しく説明します。

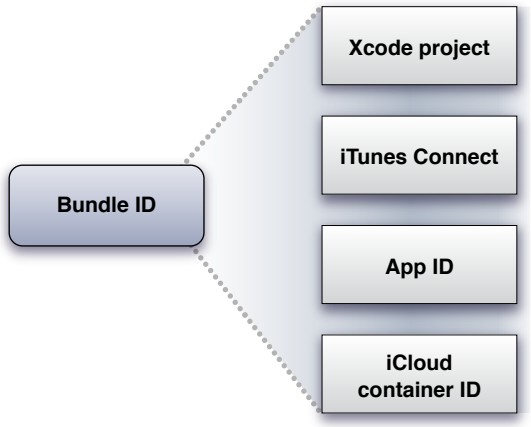
バンドルIDはアプリケーションを一意に識別する

バンドルIDは、個々のアプリケーションを厳密に特定するために用いる文字列です。開発中だけでなく、アプリケーションをデバイスにインストールした後も、オペレーティングシステムが利用します。たとえば環境設定システムは、この文字列を使って、設定をどのアプリケーションに適用するか識別します。これに対してLaunch Servicesは、あるファイルを開くために使うアプリケーションの検索にバンドルIDを利用します。IDに合致する、最初に見つかったアプリケーションで開くのです。さらにiOSも、アプリケーションの署名を検証する際、バンドルIDを使います。

バンドルIDの文字列は、英数字（`A-Z`、`a-z`、`0-9`）、ハイフン（`-`）、ピリオド（`.`）のみで構成されるUTI（Uniform Type Identifier）でなければなりません。全体はいわゆる逆DNS形式の文字列です。たとえば、会社のドメインが`Ajax.com`で、Helloという名前のアプリケーションを作成する場合、バンドルIDは「`com.Ajax.Hello`」という文字列になります。

開発中はさまざまなところで、バンドルIDを使ってアプリケーションを識別します。図 3-2にその代表的な箇所を示します。

図 3-2　アプリケーションのバンドルIDを使う代表的な箇所



- バンドルID自身は、プロジェクトの情報プロパティリストファイル（`Info.plist`）に格納されています。このファイルは、ビルドの際、アプリケーションバンドルにコピーされます。
- アプリケーションを公開する際には、「iTunes Connect」でアプリケーションを特定するために使います。App Storeの登録手続きでは、アプリケーションから抽出したバンドルIDを使って、「iTunes Connect」に登録されたデータと対応づけるようになっています。詳細については「iTunes Connectでアプリケーションデータを設定する」（42 ページ）を参照してください。
- 開発中は、バンドルIDに対応するアプリケーションIDを記述した、開発用プロビジョニングプロファイルが必要です。
- iCloudを操作する機能を実装する場合、指定するコンテナIDは、アプリケーションのバンドルIDをもとに生成したものでなければなりません。

アプリケーションIDはアプリケーションをサービスや開発チームと対応づける

アプリケーションIDも文字列で、一部はバンドルIDと共通になっています。しかし重要な違いが2つあります。まず、アプリケーションIDは、特定のアプリケーションではなく、ある一群のアプリケーションを識別するために使うこともできます。また、アプリケーションと開発チームを対応づける情報が追加されます。バンドルIDは、先頭に会社IDを置いた逆DNS形式で表す規約になっていますが、これは強制ではなく、また、開発チームの識別には使えません。アプリケーションIDは主として、開発用、配布用のプロビジョニングプロファイル（詳しくは後述）を作成する際に使います。

アプリケーションIDは、**バンドルシードID**と**バンドルID照合文字列**の2つの部分を、ピリオド（.）をはさんで連結した文字列です。この2つの部分は、それぞれ重要な用途があります。

バンドルシードIDはAppleが生成する10文字の一意的な文字列です。アプリケーションIDを開発チームに対応づける役割があります。バンドルシードIDが同じアプリケーションは、ユーザ名、パスワードなどのキーチェーンデータも共通です。開発チームには、**チームID**という特別なシードIDが割り当てられます。特段の理由がない限り、新規に開発するアプリケーションでは、バンドルシードIDとしてこのチームIDを使うようにしてください。Mac OS X用アプリケーションの場合、それ以外の選択肢はなく、ツールが自動的にそのように設定します。

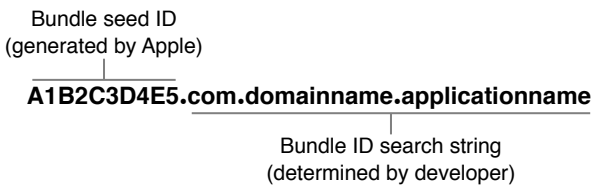
> **注意**　これまでにiOS用アプリケーションを開発したことがあれば、配布済みのアプリケーションに使ったバンドルシードIDがあるかも知れません。新しいシードIDを生成する機能はツールから削除されていますが、チームIDの代わりに、前回のバンドルシードIDをそのまま使うことは可能です。ただしこれは、既存のキーチェーンデータを、新しいアプリケーションでも共有したい場合に限ります。このような事情がなければチームIDを使ってください。

アプリケーションIDには、*ワイルドカード（wildcard）アプリケーションID*と*特定（explicit）アプリケーションID*の2種類があります。違いはバンドルID照合文字列の部分にあります。

特定アプリケーションIDは単一のアプリケーションに合致する

特定アプリケーションIDは、照合文字列の部分が、アプリケーションのバンドルIDと正確に一致します。バンドルIDをそのまま使っているので、該当するアプリケーションだけにしか合致しません。

図 3-3　特定アプリケーションID



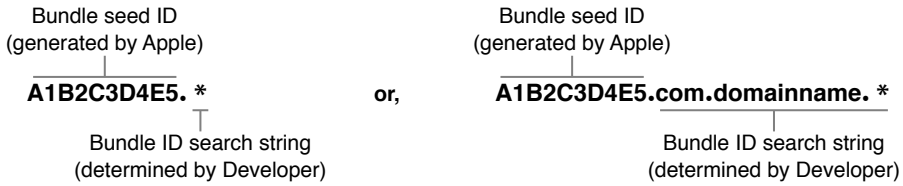
Important Apple Push Notification Service（APNS）、In-App Purchase、Game Centerなどの機能をアプリケーションに組み込むためには、特定アプリケーションIDが*必須*です。

ワイルドカードアプリケーションIDは複数のアプリケーションに合致する

ワイルドカードアプリケーションIDは、複数のアプリケーションで共通に使えます。アプリケーションごとに用意する必要はなく、1つ目のアプリケーションの開発を始める際に作成します。ただし、APNS、In-App Purchase、Game Centerの機能を組み込む場合は使えません。

ワイルドカードアプリケーションIDは、照合文字列が、バンドルIDの一部または全部をアスタリスク（*）に置き換えたものになっています。アスタリスクは照合文字列の末尾に置かなければなりません。

図 3-4 ワイルドカードアプリケーションID



アスタリスクよりも前の部分は、（特定アプリケーションIDと同様、）バンドルIDに正確に合致しなければなりません。アスタリスクはバンドルIDの残りの部分に必ず合致します。バンドルIDの1文字以上に合致しなければならないことに注意してください。図 3-5に、照合文字列の例と、どのようなバンドルIDに合致するか/しないかを示します。

図 3-5 ワイルドカードの照合例

com.domain. *		(bundle id search string)
com.domain.text	✓	* matches text.
com.domain.icon	✓	* matches icon.
com.otherdomain.database	✗	The d in the pattern fails to find a match.
com.domain	✗	The . in the pattern fails to find a match.
com.domain.	✗	The * in the pattern fails to match a character.

アプリケーションIDの照合文字列としてアスタリスク（*）だけを指定すれば、あらゆるバンドルIDと合致します。

プロビジョニングプロファイルはアプリケーション起動時の認証に用いる

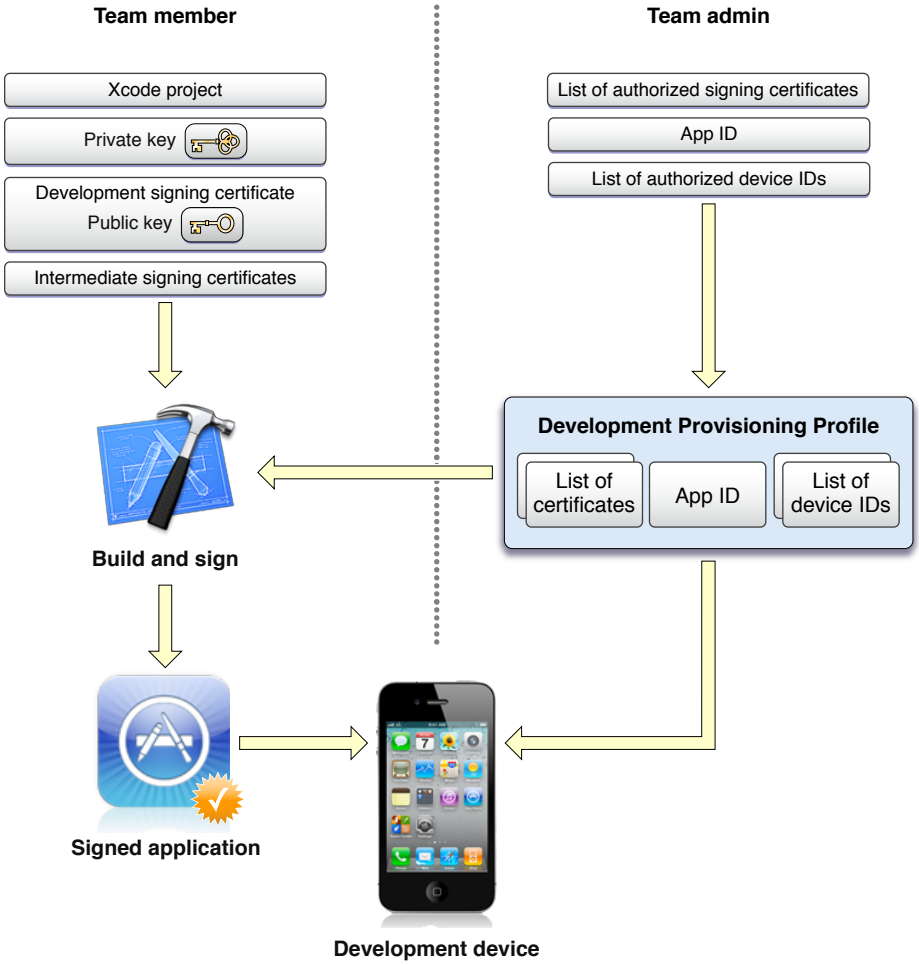
「コード署名を行う開発者の設定」（20 ページ）では、署名証明書を作成し、これを使ってアプリケーションに署名する手順を説明しました。ここでは、プロビジョニングプロファイルを使って、アプリケーションの起動を認証する仕組みを説明します。

プロビジョニングプロファイルには次の2種類があります。

- 開発用プロファイルは、開発用に設定されたデバイス上でのみ、アプリケーションの起動を認証するために使います。
- 配布用プロファイルは、アプリケーションの配布に使います。配布方法によっていくつかの種類があります。

図 3-6に、iOS用アプリケーションの開発プロビジョニングプロセスを再掲します。

図 3-6　開発プロビジョニングプロセスの概要



デバイスと署名証明書を開発チームに追加すると、Xcodeは自動的に、「*iOS Team Provisioning Profile*」という名前のプロファイルを生成します。このプロファイルは、デバイスや署名証明書を追加する都度、自動的に更新されます。このプロファイルには重要な特性が3つあります。

- チームに関係するコード署名証明書がすべて収容されています。
- チームに関係するデバイスがすべて記載されています。
- アプリケーションIDのバンドルID照合文字列はアスタリスクになっています。

したがって、iOSチームプロビジョニングプロファイルの恩恵により、チームメンバーであれば*誰*でも、*どの*アプリケーションにでも、この署名証明書を使って署名し、*どの*チームデバイスにでもインストールできることになります。「iOS Team Provisioning Profile」は自動更新されるので、チームに属するプログラマが、サンプルコードや簡単なテストアプリケーションを開発用デバイスにインストールするためには、非常に便利です。一方、App Storeで公開するアプリケーションを開発する場合は、代わりに開発用プロビジョニングプロファイルを作成する方がよいでしょう。そうすれば、次の3つの観点から制約を施し、セキュリティ保護の効果を高めることができます。

- アプリケーションIDを指定して、起動できるアプリケーションを制限できます。特定アプリケーションIDを使えば、唯一のアプリケーションしか起動できなくなります。
- アプリケーションに署名する開発者を制限できます。
- アプリケーションが動作するデバイスを、プロファイルがインストールされているものだけに制限できます。

アプリケーションを配布するためには配布用プロファイルが必要である

iOS用、Mac OS X用のアプリケーションが配布できる状態に仕上がった段階で、チーム編成時に作成した配布用署名証明書に加え、アプリケーション起動時の認証に用いる配布用プロファイルが必要になります。特にiOS用の場合、アプリケーションに署名し、適切なプロファイルと組にして配布しなければ、開発用以外のデバイスにはインストールできません。

配布用プロファイルの内容は、開発用のそれとは違っており、さらに、同じ配布用でも種類によって内容は若干異なります。最も、コード署名にチームの配布用署名証明書を使わなければならない、という要件は共通です。とは言っても、App Store上で公開する際に用いる配布用プロファイルでは、たとえば特定のデバイスでのみ動作するよう制約することはできません。

より詳しい資料

iOS用アプリケーションの開発チームに属している場合は、配布用プロファイルの種類や、新しいプロジェクトの設定方法に関して、以下の資料を参照してください。

- チーム管理者：『*iOS Team Administration Guide*』。

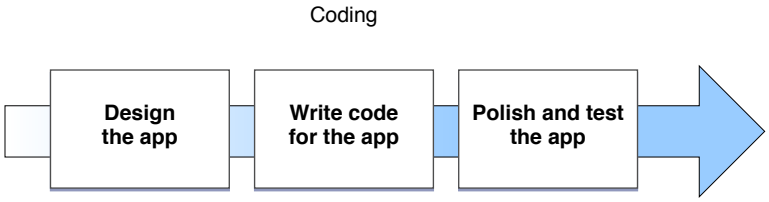
- チームメンバー：『*Tools Workflow Guide for iOS*』。

Mac OS X用アプリケーションの開発チームに属している場合は、配布用プロファイルの種類や、新しいプロジェクトの設定方法に関して、『*Tools Workflow Guide for Mac*』を参照してください。

アプリケーションの開発

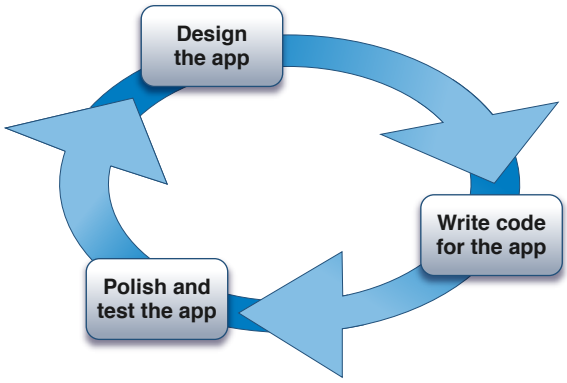
図 4-1に典型的な開発プロセスの流れを示します。アプリケーションが何をどのように実行するか、を設計し、コードの形で実装し、想定通りに動作するかテストする、という3つのフェーズです。

図 4-1　開発プロセス



上の図は理想的な開発プロセスを表しますが、現実は反映されていません。実際にはむしろ、図 4-1のようになるはずです。

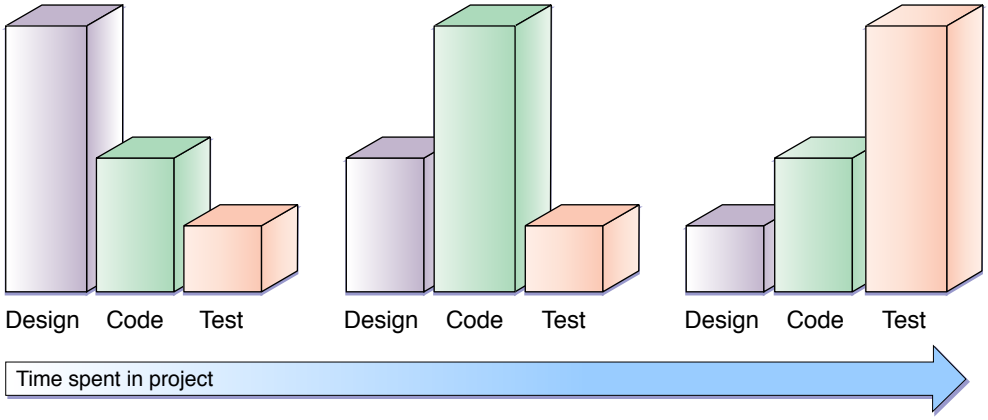
図 4-2　反復型の開発プロセス



繰り返される各回の開発プロセスにどの程度の時間を費やすかは、開発者の方針次第です。たとえば、各パスではごく少数の機能を実装するという、アジャイル開発では一般的な方針が考えられます。あるいは、全体像を記述した設計仕様を書き上げた上で、一気に実装する方針もあります。もちろん、この2つを折衷した形態もあるでしょう。いずれにしても、設計の細部にわたって繰り返し見直し、磨きをかけることが、ユーザにとって使いやすいアプリケーションに仕上げるためには重要です。ユーザにとって理解しやすいインターフェイスを提供するべきなのですが、実際にユーザに使ってもらい、その様子を観察することなしに達成するのは困難でしょう。

開発プロセスを繰り返していくにつれ、3つのフェーズの重みづけも変わっていきます。たとえば、プロジェクトを開始してすぐの段階では、アイディアはあっても、実装あるいはテストするコードがないので、設計上の意思決定に時間を費やすことになります。一方、出荷目前の段階になれば、設計はほとんど終わっており、コードを修正するのも問題が見つかった場合だけです。想定通りに動作するか、ユーザに気に入ってもらえるか、のテストに多くの時間を費やします。

図 4-3 開発作業に費やす時間配分の変遷



アプリケーション開発が初めてであれば、対象オペレーティングシステムに応じたガイドを読んでください。利用可能な技術、その上で動作するアプリケーションのアーキテクチャ、ユーザに好まれる外観や動作などの説明があります。以下に、開発時に考慮すべき重要な考え方を、上記3つのフェーズに分けて概説します。

アプリケーションの設計

アプリケーション開発が始まったら、適応性の高い設計を心がけてください。今回が初めてで、優れたアプリケーションとはどのようなものか手探り状態であれば、特にこの心がけが大切です。あるいは、最初のプロジェクトは出荷を目標とせず、技術をより深く理解するための試作と割り切って考える方がよいかも知れません。適応性の高いアーキテクチャで設計していれば、テスト中や出荷後にユーザから要望その他があったとき、迅速に対応できます。以下、最初の設計に役立ついくつかの技法を紹介します。

「モデル-ビュー-コントローラ」パターンの活用

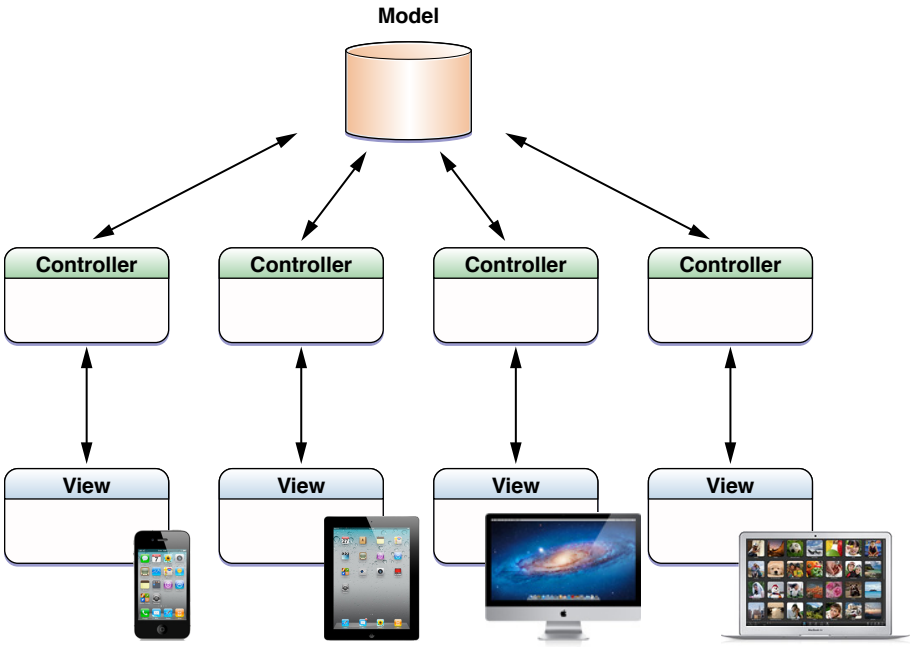
「モデル-ビュー-コントローラ」パターンは、CocoaやCocoa Touchで広く利用されているデザインパターンのひとつで、iOS用、Mac OS X用のアプリケーション設計には必須です。アプリケーションを次の3つの部分に分けて設計し、通常はそれぞれをクラスとして実装します。

- **モデル**：どのようなデータを、どのような形で、どこに格納するか。

- **ビュー**：ユーザインターフェイスはどのような外観か。どのような画面構成にするか。どのような動きをつけるか。
- **コントローラ**：ユーザはどのような処理ができるか。ユーザインターフェイスを操作したとき何が起こるか。モデルとビューが互いに連携するためにはどのような処理が必要か。

このように分割して設計すれば、ある部分を設計し直す場合でも、ほかの部分は大幅に変更せずに済みます。ユーザインターフェイスを再設計するような状況では、この考え方は重要です。理想的にことが運べば、モデルの実装を変更することなく、ユーザインターフェイスだけを置き換えることができるでしょう。そうなれば、ユーザからの要望等に応じてユーザインターフェイスを調整しやすくなるばかりでなく、アプリケーションを拡張して新しいユーザインターフェイスを提供する、あるいは複数のアプリケーションでモデルクラスを共有することも容易です。「図4-4」パターンは、iOS用、Mac OS X用のアプリケーションに共通のデータモデルを表します。

図 4-4　「モデル-ビュー-コントローラ」パターン



図のiOS用アプリケーションは、iPad向けおよびiPhone向けのユーザインターフェイスイディオムに対応しており、一方Mac OS X用アプリケーションは、ウインドウを用いたインターフェイスと、全画面に表示するインターフェイスの両方に対応します。共通のクラス群でデータモデルを表現するという設計方針は、iCloudに対応する場合にも理想的です。データをiCloudに格納する際のファイル形式は共通である一方、画面表示などのプレゼンテーション機能は、各アプリケーションがそれぞれ独自に実装します。したがって、各デバイスの特性に合わせて、理想的なインターフェイスを実現できるのです。

データ駆動設計の活用

動作をすべてソースコード中にハードコーディングすることは避けてください。処理ロジックの一部をデータとして分離する、という設計技法があります。実際の動作が、実行時に、データに基づいて決まるようにする方法です。このパターンはCocoaやCocoa Touchで頻繁に使われています。その例をいくつか示します。

- **ローカライズ：**ローカライズが必要な文字列は、多くの場合、各言語のテキストをファイルに格納しておき、実行時に読み込んで使います。こうしておけば、画面上に現れる言語を容易に切り替えることができます。ローカライズ設定を変更してアプリケーションを再起動すれば、自動的に新しい言語で表示されるのです。別の言語による新たなインターフェイスも、ソースコードを変更することなく実装できます。
- **Nibファイル：***nib ファイル*という特別な形式のファイルに、オブジェクトをデータとして格納しておくことができます。各nibファイル内のデータは、Objective-Cのオブジェクト（クラスを含む）およびその内部状態を、そのままの形で記述したものです。内部状態としてほかのオブジェクトへの参照を記述し、すべてを接続した状態で、*オブジェクトグラフ*として格納できます。開発時にnibファイルを作成してプロジェクトに保存しておくと、Xcodeはビルドの際に、これをアプリケーションバンドルにコピーするようになっています。アプリケーションの実行時には、nibファイルに格納されたオブジェクトの新たなコピーが必要になった時点で、そのインスタンスが生成されます。このときオペレーティングシステムは、nibファイルに格納されたデータを抽出し、オブジェクトを再生成し、その結果できあがったオブジェクトグラフを、すでにあるほかのオブジェクトと接続します。したがって、プログラム上でオブジェクトを生成し、必要な設定をした場合と同じ状態になるのです。

  nibファイルには多くの場合、ユーザインターフェイスオブジェクトや、関連するコントローラその他のオブジェクトを格納します。その内容は、Xcodeに付属のグラフィックツールで作成、修正できます。ユーザインターフェイス要素がデータとして格納されているので、多くの場合、ソースコードを変更することなく、独立に定義や修正が可能です。nibファイルの内容を最初に設計し、アプリケーションコードと結びつけるのはプログラマの役割ですが、コードとは分離しているので、意匠設計に長けたプログラマ以外の人が磨きをかけることも容易です。また、言語やロケールに応じて実行時に使い分けることにより、ローカライズにも活用できます。
- **ストーリーボード：**iOS 5では*ストーリーボード*がnibファイルと同様の役割を果たし、Cocoa Touchアプリケーションのユーザインターフェイス設計ではこの方法が推奨されています。もちろんnibファイルと同様の利点があります。独自のツールで作成してプロジェクトに格納しておけば、ビルドの際、アプリケーションにコピーするようになっています。実行時に読み込まれる点もnibファイルと同じです。しかしnibファイルとは違って、複数の画面にわたるデータを、単一のストーリーボードに格納できます。さらに、実行時の画面遷移を定義できる点も違います。画面群とその遷移を包括的に作成、修正可能であり、ユーザインターフェイスがどのような構成になるか、可視的に確認しながら開発できるのが特長です。ストーリーボードで管理する個々の画面は、Xcodeのツールを使って効率的に設計できます。これは複数のnibファイルを組み合わせて設

計する場合と同様です。ストーリーボードの具体的な実装については、ここでは詳しく説明しません。重要なのは、ストーリーボードの形でデータを作成しておけば、実行時に読み込まれる、という点です。

データ駆動方式のプログラミング技法を取り入れれば、クラスの設計にも違いが出てきます。たとえばゲームの場合、登場する要素（人物など）の特性を定義するために、データファイルを使う方式が考えられます。CocoaやCocoa Touchには、テキスト形式（人が読める形式）でこのようなデータを格納する、アーカイブクラスというものがあります。意匠設計に長けた人が、このファイル上でゲームの外観に磨きをかけることも可能で、もちろんコードを再コンパイルする必要はありません。通常、このようなデータはアプリケーションバンドル内のファイルに格納するのが簡単ですが、ネットワークサーバ上に置いて、随時読み込み直せるようにすることも可能です。開発の際にも、アプリケーションをいちいち終了せずにゲームデータを読み込み直すことができれば、作業効率が上がるでしょう。

以上のように、起動時に1度だけ読み込むか、再読み込みを可能にするか、の違いはあるにせよ、コードを再コンパイルせずに動作を変更できるので、ユーザインターフェイスや動作を仕上げる作業も迅速になります。

コードの記述

コードの記述に当たり、活用しようとしている技術に関するプログラミングガイドがあれば、最初に目を通しておいてください。フレームワークに属するさまざまなクラスは、どのように組み合わせて使うことを想定したものなのか、背景となる考え方が説明されています。もちろんヘッダファイルやリファレンスでも、フレームワークの動作を調べることは可能です。しかし、その設計の考え方や、アプリケーションにこの技術を組み込むための構成法については、プログラミングガイドを読むのが手っ取り早いでしょう。

Objective-Cは柔軟性の高い言語です。この柔軟性や基底にあるCのアーキテクチャが逆に災いして、アプリケーションがクラッシュしたり、想定通りに動作しなかったりしやすい、という面もあります。もちろん、問題のあるコードがいつまでも残っているのは望ましくないし、ソースコードリポジトリにチェックインすることはなおさら避けたいでしょう。そのためにも、以下の方針を取り入れるようお勧めします。

警告をエラーと見なすこと

コンパイル時に警告が現れるコードは、想定通りに動作しない可能性が高いと言えます。致命的なエラーとは表示されなくても、実際にはそれに匹敵する重大な問題が隠れているかも知れません。むしろ、警告はエラーと見なすようXcodeプロジェクトを設定し、ほかのコンパイルエラと同じように対処してください。

Xcodeのアクション分析機能を使って、隠れたコーディングエラを検出すること

Objective-Cのコンパイラには、よくあるコーディングエラーを検出する、コードアナライザが組み込まれています。Cの構文規則に基づくエラーだけでなく、CocoaやCocoa Touchのコーディング規約に反する記述にも対応しています。特別な条件でしか顕在化しない論理エラーやコーディング規約違反は、問題が起こる兆候ととらえ、その箇所を指摘します。リポジトリにチェックインする前に、必ずAnalyzeコマンドでチェックするようにしてください。

ユニットテスト、ユーザインターフェイステストを実施すること

ユニットテストは、（人手によらずに）アプリケーションをテストするコードの一種です。クラスがどのように動作するべきかは、そのクラスを実装した人がよく理解しているはずなので、想定通りに動作するかどうかのテストも記述するとよいでしょう。典型的なユニットテストでは、アプリケーション中のクラスからインスタンスを生成し、何らかの設定した後、メソッドを呼び出して実際の処理を行います。最後に、アサーションとして、想定通りの結果が得られたかどうかを調べます。Xcodeにはユニットテストの支援機能が組み込まれています。プロジェクトを作成する際、自動的に、ユニットテストをビルド、実行するターゲットを設定できます。必要なのはテスト用のコードを記述することだけです。

一貫して定期的にユニットテストを実施することにより、ソースコードを変更する際に、新たな問題が入り込む危険を抑えることができます。ユニットテストでは検出できないバグが見つかった場合は、新たにテストを記述する必要があることを示しています。クラスの動作を検証できるばかりでなく、クラスの設計自体を改良する効果もあります。オブジェクトをそれだけ切り離してテストするためには、ほかのオブジェクトやクラスに大きく依存することなく、当該オブジェクトのインスタンスを生成できるようにしなければなりません。場合によっては、テスト対象オブジェクトの処理の一部を抜き出し、そこだけをオーバーライドできるようにする必要があります。アプリケーションのほかのサブシステムを呼び出す代わりに、テスト専用のスタブコードを使ってテストするためです。このように適応性の高いクラス設計を心がければ、きれいなインターフェイスを保てるでしょう。

ユーザインターフェイスを担うクラスはユニットテストが困難です。ユーザの操作をスクリプトとして記述し、自動化するツールが必要かも知れません。iOS用アプリケーションの場合は、Instrumentsアプリケーションの自動化ツールが利用可能です（『*Instruments User Guide*』を参照）。

ユニットテストの活用方針によって、開発プロセスのどの段階で記述するかも異なります。たとえば、一般的なアジャイル方式の開発（**テスト駆動開発**）では、最初に失敗するテストを記述し、それからコード本体を追加するよう推奨しています。最初に失敗するテストを作成するのは、コードを記述する前に、テスト自体が正しいことを確認するためです。テスト駆動開発の場合、開発が進むにつれてテストスイートも増えていき、クラスのインターフェイスや実装を変更するごとにリファクタリングされていきます。もちろん、これ以外の方針も考えられます。たとえば、どのような設計や実装がよいか実験しながら開発を進める場合、最初のうちはクラスのインターフェイスを固定しないかも

知れません。この場合、クラスの設計がある程度固まってから、テストを設計する方がよいでしょう。ただし、テストを実装する時点で、クラス自体のインターフェイスや実装を若干変更しなければならないこともあります。

アプリケーションの仕上げとテスト

コードを実装した後、テストを実施して、バグその他、想定通りでない動作を除去していきます。多くのユニットテストを実施し、さまざまなツールを使っていても、バグは入り込んでしまうものです。しかしそれ以外にも、テストしなければならない事項があります。アプリケーションの動作が、ユーザにとって満足できるものであるか、という点もそのひとつです。小気味よく振る舞い、関心をつなぎ止めるようなアプリケーションが望ましいことは、言うまでもありません。

アプリケーションの仕上げ段階では、次のガイドラインを考慮してください。

さまざまなデバイス上でテスト

デバイスによってその能力はさまざまです。各種のデバイス上でテストすれば、アプリケーションの動作に関して、見当外れの想定をしないで済みます。これはiOS用アプリケーションの場合に特に重要です。開発中はシミュレータも便利ですが、実際のハードウェアの性能特性やメモリ量の制約に関しては、シミュレートできません。実際のデバイス上にアプリケーションをインストールし、テストしてください。

アーカイブアクションによりテスト版アプリケーションを作成

Xcodeのアーカイブアクションを使うと、アプリケーションをビルドする際、デバッグ用の情報を、Xcodeが管理するバンドルに格納できます。アーカイブはアプリケーションの配布時に使いますが、開発中も便利です。たとえば、テスト版をテスト担当者に送る際、デバッグ情報を組み入れておけば、クラッシュレポートの分析が容易になります。アーカイブを作成する段階になったら、ビルドやコンパイラに関するほかのプロジェクト設定も見直すとよいでしょう。

> **Important**　ユーザ向けに配布するアーカイブは、すべての版を保存しておいてください。クラッシュレポートを分析するためには、アーカイブ内のデバッグ情報が必要です。

Instrumentsで動作状態を確認

Xcodeに付属のInstrumentsアプリケーションは、Profileアクションを通して容易に利用できます。メモリ使用状況、オペレーティングシステムのほかのサブシステムの使用状況、負荷が高い状態での動作などを調べるために不可欠です。iOS用アプリケーションの場合は、Instrumentsで電池の消費量も調べることができます。

認定ガイドラインに従うこと

アプリケーションをアーカイブ化し、公開の準備が整ったら、XcodeのTestアクションで検証してください。これは、見つかれば即座に認定を却下されるような、よくある問題点を洗い出すツールです。この自動化されたテストに加え、ガイドラインを読み返し、それに従っているかどうか確認することも大切です。

優れたアプリケーションの開発

優れたアプリケーションの開発は、技術と言うよりもむしろ芸術と考えるべきでしょう。しかし品質向上という観点からは、技術が果たす役割も多分にあります。特に、陥りやすい落とし穴を避けることにより、ユーザに好まれないアプリケーションになってしまうのを避けることができます。満足したユーザはApp Storeで高評価をつけ、知り合いにも勧めるに違いありません。

設計や実装の段階で考慮するべき事項をいくつか紹介します。

ユーザ指向のアプリケーション設計

アプリケーションの設計に当たっては、ユーザにとっての使いやすさを重視してください。必要もないのに処理が中断したり、作業を妨げたりする状況は避けなければなりません。

デバイスの特性を考慮し、そのデバイスを使うユーザの期待を裏切らないインターフェイスになるよう工夫してください。たとえばiPhoneは、ほかに比べて画面が小さいばかりでなく、操作形態にも大きな違いがあります。ポケットから取り出してちょっとした作業をし、すぐにまた戻す、というように、1回当たりの操作時間が短いのが特徴です。一方iPadは画面が大きく、ある程度長い時間、連続して使う傾向があります。Macも画面は大きく、長時間にわたって使うことが多いでしょう。iPhone用アプリケーションを設計する際には、1回当たりの操作時間が短いので、処理が中断すれば、iPadやMac OS Xの場合に比べていらいらが増す、ということを考慮してください。OSやデバイスの特性に合わせて調整し、Appleのヒューマンインターフェイスガイドラインに従えば、ユーザの満足度を上げることができるでしょう。

セキュリティモデルの開発

昨今は特に、あらゆるアプリケーションやオペレーティングシステムで、セキュリティの重要性が増しています。デバイスは常にネットワークにつながっています。攻撃を受けたデバイスが、ほかのデバイスを攻撃する踏み台として使われる可能性もあります。デバイスの制御を奪う攻撃ばかりではありません。デバイスには大切な個人情報が格納されているかも知れないのです。紛失や盗難に遭えば、データの安全性に懸念が生じるでしょう。

- **データの流れに着眼**：アプリケーションがデータをどのように受け取り、格納し、転送し、使うかを検証してください。たとえば、iOSデバイス上のファイルとしてデータを格納しているとすれば、iTunesにバックアップする際に、盗聴され、あるいは改変される恐れがあります。このデータを再び取り込んで同期し、アプリケーションで読み込めばどうなるでしょうか。あるいは、ネットワーク経由でデータを受け取る場合、ほか人から受け取ったデータは悪意ある攻撃に使えるよう、巧妙に仕組まれているかも知れません。アプリケーションを通してデータがどのように流れるか（そしてどの程度の信頼レベルか）を理解することが、さらされうる攻撃を予測する上で重要です。危険を予測できれば、攻撃を検出し回避するために役立つでしょう。
- **作業実行に必要な最低限の権限のみ取得**：許されていない処理までできてしまうアプリケーションは、悪用される恐れがあり危険です。Mac OS Xで、高レベルの権限を要する処理を実行する場合は、アプリケーションをいくつかのプロセスに分割した上で、当該処理を実行するプロセスにのみ、必要最低限の権限を与えるようにしてください。

  この原則が当てはまる別の例としてサンドボックスがあります。あるアプリケーションが格納したデータファイルに、ほかのアプリケーションがアクセスできないようにする仕組みです。iOS用アプリケーションは自動的にサンドボックス対応になります。Mac OS Xの場合は、明示的にサンドボックス処理を実装してアプリケーションを保護しなければなりません。
- **処理失敗時のセキュリティ維持**：処理に失敗した場合でも、デバイスやユーザデータのセキュリティは維持できるよう、安全側に倒して実装してください。

アプリケーションの信頼性確保

アプリケーションの欠陥は可能な限りなくするべきです。とは言え、現実の世界では、思わぬところでエラーが起こるものです。オペレーティングシステムがエラーを返した場合でも、適切に対処するよう設計、実装しなければなりません。いつでも起こりうると想定して備えるべき、一般的なエラーをいくつか示します。

- **ネットワークの停止**：モバイルデバイスは、ネットワークハブや携帯電話中継局から離れて移動すると、頻繁にネットワークが切り替わったり、接続が切れたりします。ネットワーク接続が必要なアプリケーションの場合、処理中であっても切断や切り替えが起こりうると想定しておかなければなりません。
- **ストレージの容量不足**：デバイスの容量には上限があります。iCloudも、保存できる最大容量が決まっています。データを書き込もうとして、その限度を超えてしまう危険は常にあります。
- **メモリ不足**：デバイスのメモリ容量にも上限があります。iOSでは、メモリが不足し、システムからの解放要求に応じられなければ、強制終了させられてしまいます。Mac OS Xでは、メモリ消費量が増えても処理は続行できますが、仮想メモリを大量に使うため非常に遅くなります。低メモリ下での動作も考え、キャッシュその他のリソースを解放できるよう用意しておかなければなりません。

アプリケーションの設計時には、常に*フェイルセーフ*であるように検討してください。自分が開発した部分のプログラミングエラー、あるいは避けられない条件（メモリが確保できず、解放もできないなど）により、アプリケーションが停止してしまうかも知れません。ユーザが意図的に停止することも考えられます。そのような場合でも、データの破損だけは避けなければなりません。フェイルセーフを実現する方法の一例として、あらゆる処理を、既存のデータを直接書き換えずに行う、というものがあります。処理が成功したことを確認してから、その結果と、従来の中身とを交換するのです。この技法はファイルをストレージに書き出す際によく使われます。ファイルに直接上書き保存するのではなく、別のファイルにいったん書き出した後で、旧ファイルと差し替えることになります。

アプリケーションの性能改善

アプリケーションの体感性能は、ユーザに気に入ってもらえるかどうかを決める重要な因子です。起動画面でハングしたり、入力に対しておかしな応答をしたりすれば、印象は最悪です。さらに、性能が劣るアプリケーションの多くは*非効率*でもあります。モバイルデバイスの場合、このようなアプリケーションは電池の消耗が激しく、1回の充電で使える時間が少なくなってしまいます。

開発に当たっては、次に挙げる事項を念頭に置いてください。

- **迅速な応答が必要な機能を把握する**：たとえばアプリケーションの起動時間は、少しでも短くなるよう、開発プロセスを通して常に心がけなければなりません。もちろんほかの重要な処理も、常に迅速に実行する必要があります。時間がかかる場合は、その旨がユーザに分かるよう、何らかの目印を示すべきでしょう。このような処理のために、アプリケーションが応答しなくなってしまっては困ります。バックグラウンドで処理するようにしてください。
- **解決しようとする問題に合わせて実装する**：使用するクラス（あるいは実装するアルゴリズム）は、処理内容に合わせて選ばなければなりません。たとえば、処理対象が数十件程度の場合と数百万件に及ぶ場合では、大きな違いがあります。負荷を考慮して適切に設計にしてください。
- **時期尚早な最適化を避ける**：あまりにも複雑なアルゴリズムや解法の実装は、性能を考え、どうしても必要と判明するまで避けてください。むしろ、読んで理解しやすい設計を心がけるべきでしょう。クラスのインターフェイスを簡明に定義し、実装詳細はその内部に隠して、ほかのクラスに依存することなく実装を変更できるようにしてください。こうしておけば理論上、ユニットテストに合格する限り、いつでもクラスの実装をより効率的なものに変更でき、ほかの部分に影響を与えることもありません。
- **どの部分で時間を費やしているか、実際に計測して判断する**：予想外の箇所で処理時間を費やしていることも少なくありません。きちんと把握するためには、プロファイルを取るというのが最も信頼できる方法です。Instrumentsにはさまざまなプロファイリングツールがあるので、アプリケーションの実際の動作を把握するために役立つでしょう。

より詳しい資料

iOS用アプリケーションを開発する場合は、以下の資料を参照してください。

- iOS用アプリケーションの開発について：『*iOS App Programming Guide*』。
- iOS用アプリケーションに組み込むことができる技術について：『*iOS Technology Overview*』。
- iOSのユーザインターフェイス規約に則ったアプリケーションの開発について：『*iOS Human Interface Guidelines*』。
- アプリケーション開発に必要な作業について：『*Tools Workflow Guide for iOS*』。

Mac OS X用アプリケーションを開発する場合は、以下の資料を参照してください。

- Mac OS X用アプリケーションの開発について：『*Mac App Programming Guide*』。
- Mac OS X用アプリケーションに組み込むことができる技術について：『*Mac OS X Technology Overview*』。
- Mac OS Xのユーザインターフェイス規約に則ったアプリケーションの開発について：『*Mac OS X Human Interface Guidelines*』。
- アプリケーション開発に必要な作業について：『*Tools Workflow Guide for Mac*』。

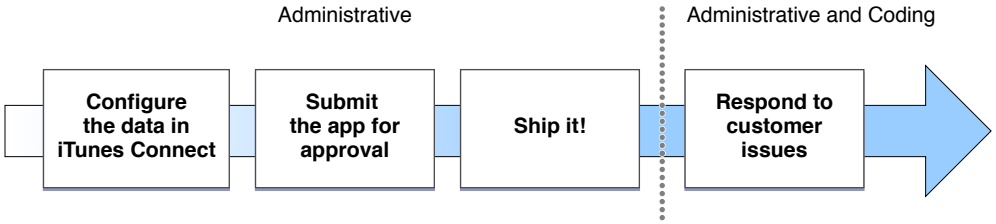
アプリケーションの設計や実装を効率よく進めるための技法

コードの品質を向上するため、以下の資料を参照してください。

- CocoaおよびCocoa Touchのコーディング規約について：『*Coding Guidelines for Cocoa*』。自分が記述するソースコードにも同じ規約を適用すれば、一貫性が保てます。
- セキュリティを高める方法について：『*Secure Coding Guide*』および『*Security Overview*』。
- アプリケーションの効率を改善する方法について：『*Performance Overview*』。

アプリケーションをApp Storeで公開

バグがなくなれば、いよいよ公開です。App Storeでの準備に取り掛かります。



iTunes Connectでアプリケーションデータを設定する

App Storeの画面には、アプリケーションの名前、説明、アイコン、スクリーンショット、連絡先など、さまざまな情報が表示されます。こういった情報は、iTunes Connectにログインし、アプリケーションごとに作成したレコードに入力していきます。レコードにはバンドルIDの設定フィールドもあります。アプリケーションのバンドルIDを、正確に入力してください。

Game CenterやIn-App Purchaseなど、Appleの技術の中には、開発の初期段階でiTunes Connectにレコードを作成しないと使えないものがあります。たとえばIn-App Purchaseの場合、販売する商品の詳細を入力するために、アプリケーションレコードが必要となります。これは開発プロセスが終結する前に作成して、In-App Purchaseの機能を組み込むために追加したコードをテストしなければなりません。

Important iTunes Connectレコードは開発終結までに作成しなければなりませんが、その作業はできるだけ遅らせてください。iTunes Connectレコードを要する機能の実装は、開発サイクルの最終段階で行う、ということです。というのも、アプリケーションは、iTunes Connectレコード作成後、一定期間内に公開しなければならないからです。期限を過ぎるとレコードは削除されるので、改めて作成し、データを入力し直さなければなりません。これは、節操のない開発者が、いつまでもアプリケーション名を「占有」し続けないようにするのが目的です。詳しくは『iTunes Connect Development Guide』を参照してください。

（期限の制約を避けるため）この機能を組み込まない版をいったん公開し、以降の版で実装する、という方法もあります。

アプリケーションの認定を求める

開発が終結間近になってきたら、アプリケーションのアーカイブ作成を始めてください。アーカイブは大雑把に言うと、ビルドしたアプリケーションと、対応するデバッグシンボル情報をまとめたものです。アプリケーションの認定を求める段階になると、チーム管理者はこのアーカイブに対して、次の2つの作業を行います。

- Xcodeでアーカイブを検証する。アプリケーション自身、およびiTunes Connectレコードに入力した情報が対象で、実際の検証処理は自動化されています。
- Xcodeを使ってアーカイブを登録し、認定を求める。Xcodeはアーカイブを Appleに送ります。Appleでは、ガイドラインに則っているかどうかを審査します。実行中にクラッシュしたり、iTunes Connectレコードに記載されている処理ができなかったりすると、却下になります。

却下になった場合は、指摘された問題点を修正し、再登録してください。

アプリケーションを出荷する

iTunes Connect上で、アプリケーションをApp Storeに公開する日を設定してください。認定後ただちに公開しても、将来の日付を設定しても構いません。マーケティング活動など、公開に向けて準備が必要であれば、それに応じた日付を設定するとよいでしょう。

ユーザからの要望に応える

作業はこれで終わりではありません。当然、ユーザがアプリケーションをどのように受け止めているか、気になることでしょう。App Storeにはユーザの評価や論評が載るため、アプリケーションを成功に導くうえでおおいに参考になります。問題が指摘された場合は迅速に対処し、初公開時と同様の手続きで改訂版を登録してください。

iTunes Connectのサイトには、売上その他の財務情報、ユーザの論評、ユーザがAppleに寄せたクラッシュログなど、開発の成果を判断するために役立つデータが表示されます。クラッシュログは、ユーザのもとで重大な問題が起きていることを表すので、特に重要です。最優先で問題分析に取り組んでください。よくあるクラッシュログを以下に示します。

- **アプリケーションのクラッシュ：**「アプリケーションのクラッシュ」は、不正なメモリアクセス、例外、その他プログラミング上のエラーにより、処理が停止したことを表します。
- **メモリ不足：**「メモリ不足」の警告は、必要なメモリを確保できず、システムが強制終了させたことを表します。
- **ユーザによる強制終了：**「強制終了」メッセージは、アプリケーションが応答しなくなり、ユーザが強制終了させたことを表します。
- **ウォッチドッグタイムアウト：**「ウォッチドッグタイムアウト」は、起動や停止、システムイベントへの応答に、時間がかかりすぎていることを表します。

メモリ不足の場合を除き、クラッシュログには、停止時の各スレッドのスタックトレースが記載されています。クラッシュログはXcode Organizer上で開いてください。開発用コンピュータに、クラッシュログを生成したのと同じ版に対応するアーカイブがあれば、Xcodeは自動的に、クラッシュログに記載されたアドレスを、実際のクラスや関数に対応づけて表示します。この処理を*シンボル解決*と言います。この処理についてはワークフローガイドを参照してください。

より詳しい資料

アプリケーションの設計や実装に際して、登録して認定を求める手続きまでの間に、『App Store Review Guidelines』を読んでください。

iOS用アプリケーションの開発チームに属している場合は、アプリケーションの配布手続きに関して、次の資料を参照してください。

- チーム管理者：『*iOS Team Administration Guide*』。
- チームメンバー：『*Tools Workflow Guide for iOS*』。
- iTunes Connectおよびここで実行する作業の包括的な解説：『iTunes Connect Development Guide』。

Mac OS X用アプリケーションの開発チームに属している場合は、チームの編成、署名証明書の設定に関して、次の資料を参照してください。

- チーム管理者およびチームメンバー：『*Tools Workflow Guide for Mac*』。
- iTunes Connectおよびここで実行する作業の包括的な解説：『iTunes Connect Development Guide』。

書類の改訂履歴

この表は「*App Store での公開に向けた開発*」の改訂履歴です。

日付	メモ
2012-01-09	このドキュメントは、以前は『アプリケーション開発の概要』という名前でした。今回、ドキュメント名を変更し、新たに図版を追加し、説明を追加し、必要な修正を施しました。
2011-10-12	iOSおよびMac OS Xアプリケーションを開発してApp Storeに登録する方法について説明した新規ドキュメント。

用語解説

このドキュメントに出てくる用語を解説します。

アプリケーションID（app ID） 一意的なデジタル指紋で、アプリケーションがキーチェーンのある部分にアクセスする許可を得るために必要です。開発用/配布用プロビジョニングプロファイルの一部でもあります。

バンドル（bundle） 標準的な階層構造になったディレクトリで、実行形式コードや、これが使うリソースを収容します。

バンドルID（bundle ID） バンドルを一意に特定する文字列。

コード署名証明書（code signing certificate） デジタルIDに、名前、電子メールアドレス、会社名などの情報を関連づけた電子ドキュメント。

コード署名ID（code signing identity） コード署名証明書、および対応する秘密鍵の組。アプリケーションに署名するために使います。

開発用プロビジョニングプロファイル（Development Provisioning Profile） エンティティとエンタイトルメントの組で、開発用デバイスにアプリケーションをインストールし、テストできるようにするために使います。名前、開発用証明書のリスト、デバイスIDのリスト、アプリケーションIDが記述されています。

開発チーム（development team） Individual Programに登録された個人、またはStandard Programに登録された、チームエージェントと、複数人のチーム管理者およびチームメンバーから成る団体。

配布用プロビジョニングプロファイル（distribution provisioning profile） エンティティとエンタイトルメントの組で、アプリケーションを配布するために必要です。名前、配布用証明書、アプリケーションIDが記述されています。

開発用署名証明書（development signing certificate） アプリケーションの開発用に制限されたコード署名証明書。

配布用署名証明書（distribution signing certificate） アプリケーションの配布用に制限されたコード署名証明書。

iTunes Connect ウェブベースの開発者向けツール群。App Storeで販売するアプリケーションの登録や管理に使います。

iOS iOSは、iPhone、iPad、iPod touchのデバイス上でアプリケーションをネイティブに実行するためのOSとテクノロジーで構成されています。iOSは、共通の遺産と多くの基盤となるテクノロジーをMac OS Xと共有していますが、ユーザのニーズが若干異なるモバイル環境のニーズに対応するように設計されました。

キーチェーン（keychain） Mac OS XおよびiOSで、暗号化されたパスワード、秘密鍵など、機密情報を格納したデータベース。暗号化や認証に用いる、証明書その他秘密でない情報も格納できます。

ソースコードリポジトリ（source code repository） ソース管理システムが管理するファイルを収容するディレクトリ木あるいはデータベース。

ソース管理（source control） ファイルに対して施された変更を、時系列に沿って管理するためのツールおよび手続き群。SCM（Source Control Management）、バージョン管理などとも言います。

チームエージェント（team agent） 開発チームにおいて、最初にDeveloper Programへの入会を申請し、認められた者。Developer Program Portalのあらゆる機能を利用でき、Developer Programの契約を遵守する責任を負います。

チーム管理者（team admin） 開発チームにおいて、チームエージェントまたはほかのチーム管理者に指名され、法的契約の署名を除いてチームエージェントと同じ権限を持つ者。

チームメンバー（team member） 開発チームにおいて、開発用証明書を要求し、開発用デバイスにアプリケーションをインストールすることができる者。

テスト駆動開発（TDD、test-driven development） （失敗する）テストケースをまず記述してから、プロジェクトのコードを書き換える、というプログラミング技法。テストケースを記述し、それが失敗することを確かめた後に、新しいテスト（および以前作成したテスト）に成功するようコードを書き換える、という手順で進めます。

一意デバイスID（UDID、Unique Device ID） あるiOSデバイスを特定するために使う40字の文字列。

ユニットテスト（unit test） アプリケーションのある部分を実行するコード。ある入力を与えて実行し、その戻り値が想定通りであるかどうか確認します。

Xcode Mac OS X用、iOS用にAppleが提供している開発環境。Mac OS X用、iOS用のアプリケーションを作成、デバッグ、最適化するためのツール群です。

Xcodeプロジェクト（Xcode project） 製品を構築するために必要な、ソースファイル、ライブラリ、メディアその他のリソースを集めたもの。

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/