**Software Requirements Specification**

**CareerCraft: AI-Powered Career Building Assistant**

Version 1.0

**Prepared by:**

Muneeb Arif

Faiqa Fatima

**Organization:** Digital Genei

**Date Created:** July 28, 2025

# Table of Contents

Contents

[Table of Contents 1](#_Toc184668502)

[Revision History 2](#_Toc184668503)

[1. Introduction 3](#_Toc184668504)

[1.1 Purpose 3](#_Toc184668505)

[1.2 Document Conventions 3](#_Toc184668506)

[1.3 Intended Audience and Reading Suggestions 3](#_Toc184668507)

[1.4 Product Scope 3](#_Toc184668508)

[1.5 References 3](#_Toc184668509)

[2. Overall Description 4](#_Toc184668510)

[2.1 Product Perspective 4](#_Toc184668511)

[2.2 Product Functions 4](#_Toc184668512)

[2.3 User Classes and Characteristics 4](#_Toc184668513)

[2.4 Operating Environment 4](#_Toc184668514)

[2.5 Design and Implementation Constraints 4](#_Toc184668515)

[2.6 User Documentation 4](#_Toc184668516)

[2.7 Assumptions and Dependencies 4](#_Toc184668517)

[3. External Interface Requirements 5](#_Toc184668518)

[3.1 User Interfaces 5](#_Toc184668519)

[3.2 Hardware Interfaces 5](#_Toc184668520)

[3.3 Software Interfaces 5](#_Toc184668521)

[3.4 Communications Interfaces 5](#_Toc184668522)

[4. System Features 6](#_Toc184668523)

[5. Other Nonfunctional Requirements 7](#_Toc184668524)

[6. Domain Model: 8](#_Toc184668525)

[7. Behavioral Modeling: 9](#_Toc184668526)

[7.1 System Sequence Diagram: 9](#_Toc184668527)

[7.2 Activity Diagram: 10](#_Toc184668528)

[7.3 State Machine Diagram: 11](#_Toc184668529)

[8. Logical Models: 12](#_Toc184668530)

[8.1 Class Diagram: 12](#_Toc184668531)

[8.2 Sequence Diagram: 13](#_Toc184668532)

[9. Operation Contracts: 14](#_Toc184668533)

[9.1 Contract UC1: LiveStreamMatch 14](#_Toc184668534)

[9.2 Contract UC2: SubscribeToMatchUpdates 14](#_Toc184668535)

[9.3 Contract UC3: ManageUserAccount 14](#_Toc184668536)

[9.4 Contract UC4: ManageContentByAdmin 14](#_Toc184668537)

[9.5 Contract UC5: AccessMatchReplayOrHighlights 15](#_Toc184668538)

[9.6 Contract UC6: JoinLiveChat 15](#_Toc184668539)

[9.7 Contract UC7: ConfigureCustomNotifications 15](#_Toc184668540)

[9.8 Contract UC8: UploadUserContent 15](#_Toc184668541)

[9.9 Contract UC9: SubscribeToPremiumFeatures 16](#_Toc184668542)

[9.10 Contract UC10: ProcessPayment 16](#_Toc184668543)

[10. Collaboration Diagram 17](#_Toc184668544)

[11. Component Diagram: 18](#_Toc184668545)

[12. Package Diagram: 19](#_Toc184668546)

[13. Deployment Diagram: 20](#_Toc184668547)

[14. Data Flow Diagram (DFD): 21](#_Toc184668548)

[15. GOF Design Patterns: 22](#_Toc184668549)

[15.1 Creational Pattern: Singleton 22](#_Toc184668550)

[15.2 Structural Pattern: Adapter 22](#_Toc184668551)

[15.3 Behavioral Pattern: Observer 23](#_Toc184668552)

[16. GRASP Patterns 26](#_Toc184668553)

[16.1 Creator 26](#_Toc184668554)

[16.2 Information Expert 26](#_Toc184668555)

[16.3 Controller 26](#_Toc184668556)

[16.4 Low Coupling 26](#_Toc184668557)

[16.5 High Cohesion 27](#_Toc184668558)

[16.6 Polymorphism 27](#_Toc184668559)

[16.7 Pure Fabrication 27](#_Toc184668560)

[16.8 Indirection 27](#_Toc184668561)

[16.9 Design Improvements 27](#_Toc184668562)

[Appendix A: Glossary 29](#_Toc184668563)

[Appendix B: Analysis Models 29](#_Toc184668564)

[Appendix C: To Be Determined List 30](#_Toc184668565)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Muneeb, Faiqa | 28/07/2025 | Initial draft of the SRS for CareerCraft | 1.0 |
| Tahmeed, Faiqa | 13/11/2024 | Adding Conceptual and Logical Models | 2.0 |
| Tahmeed, Faiqa | 12/9/2024 | Adding Communication, Component, Package, Deployment, DFD, Design and Grasp Pattern | 3.0 |

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to specify the requirements for the **CareerCraft platform**, which is designed to guide students, graduates, and professionals in building their career paths using AI-driven tools. It addresses key career development challenges such as resume creation, skill-gap identification, career roadmap planning, and mock interview simulations. This document outlines both functional and non-functional requirements necessary to design, develop, and deploy the system effectively.

## 1.2 Document Conventions

Standards for numbering and notation:

* Requirements will start with "The system should."
* All requirements are numbered for clarity.

## 1.3 Intended Audience and Reading Suggestions

This document is meant for developers, designers, testers, project managers, and stakeholders of the CareerCraft project.

* Developers should focus on technical and functional requirements.
* Designers and testers should follow UI and usability rules.
* Managers and stakeholders should review the overall scope and goals.

## 1.4 Product Scope

CareerCraft is a **web-based AI-powered platform** that assists users in crafting and advancing their careers through features such as **resume generation, skill-gap analysis, AI-driven interview simulations, personalized career roadmaps, and portfolio building**. It targets students, job seekers, and professionals—especially in underrepresented or underserved regions—by offering accessible, intelligent, and structured guidance for career growth. The platform provides a **seamless and responsive user experience**, integrates with external AI and course suggestion APIs, and allows users to track progress and showcase their growth through customizable portfolios.

## 1.5 References

* SRS Template (IEEE)
* Streaming protocols - HTTP Live Streaming (HLS), Dynamic Adaptive Streaming over HTTP (DASH)
* Regulatory standards for streaming in specific regions (GDPR, CCPA)

# 2. Overall Description

## 2.1 Product Perspective

CareerCraft is a web-based application that integrates AI and third-party APIs to support career development.  
It connects with AI services (like OpenAI), course platforms, and databases to deliver personalized career tools.  
The platform is built for responsive use and follows a modular frontend-backend architecture.

## 2.2 Product Functions

* AI-powered resume creation and improvement
* Skill-gap analysis based on job descriptions
* AI-driven mock interview simulations
* Career roadmap with course suggestions
* Portfolio creation and progress tracking
* User authentication and dashboard access

## 2.3 User Classes and Characteristics

* **General Users**: Students, job seekers, or professionals using AI tools for career development
* **Admin Users**: Manage platform content, user data, and system insights
* **Guests**: Can explore platform features but need to sign up to access tools

## 2.4 Operating Environment

The software will run on modern web browsers (Chrome, Firefox, Safari, Edge) on both desktop and mobile.  
It requires a stable internet connection to interact with AI APIs and third-party services.  
The UI is responsive and optimized for screen sizes from 5" (mobile) to 15" (desktop/laptop).

## 2.5 Design and Implementation Constraints

* The system should adhere to data protection laws (GDPR, CCPA).
* The system should integrate with third-party APIs for live match data.
* The system should work under variable internet speeds, using adaptive streaming protocols.

## 2.6 User Documentation

* The system should provide online help and FAQs.
* The system should offer in-app tutorials for new users.

## 2.7 Assumptions and Dependencies

* The system assumes reliable access to third-party APIs for match data.
* The system assumes users will have smartphones with stable internet connectivity.

# 3. External Interface Requirements

## 3.1 User Interfaces

* **Home Screen**: The system should display live and upcoming matches with quick access to the streaming page.
* **Match Streaming Screen**: The system should provide a video player with real-time event overlays (goals, fouls, substitutions).
* **User Profile Screen**: The system should allow users to manage preferences, account details, and notifications.

## 3.2 Hardware Interfaces

* The system should interact with mobile devices using standard input methods like touchscreens and notifications.

## 3.3 Software Interfaces

* The system should integrate with cloud-based streaming services and user authentication systems.
* The system should connect to APIs for live soccer match data and event notifications.

## 3.4 Communications Interfaces

* The system should use HTTPS for secure communication.
* The system should use real-time communication protocols like Web Sockets for live updates.

# 4. System Features

* The system should provide live video streams of soccer matches.
* The system should automatically adjust the video quality based on network conditions.
* The system should allow users to pause and resume live streams.
* The system should update match information within 5 seconds of real-time events.
* The system should send notifications for goals, penalties, and match completion.
* The system should support user sign-up and login via email or social media.
* The system should allow users to manage account settings, including password changes.
* The system should provide replay options for all completed matches.
* The system should allow users to watch highlights of key moments (e.g., goals, fouls) for each match.
* The system should categorize and provide access to the top-rated match highlights.
* The system should allow users to participate in live chats during match streaming.
* Users should be able to react to match events with emojis or quick reactions (e.g., goal celebrations, red card reactions).
* Users should have the option to mute the chat if they wish to avoid distractions.
* The system should allow users to select specific matches or teams to receive notifications for goals, halftime, and full-time events.
* The system should allow users to configure notifications for individual players' performance events (e.g., goals, assists).
* Notifications should be customizable in terms of alert type (e.g., sound, pop-up).
* The system should allow users to upload video or text-based match analysis.
* User-generated content should be categorized and available for other users to view and rate.
* The system should moderate, and filter inappropriate content based on predefined guidelines.
* The system should provide ad-free streaming for premium users.
* Premium users should have access to multiple camera angles and commentary options during live matches.
* The system should allow premium users to access exclusive match replays and behind-the-scenes content.
* Premium users should have priority customer support via live chat.
* The system should support payments through credit/debit cards, PayPal, and mobile wallets (e.g., Apple Pay, Google Pay).
* The system should allow users to select and save preferred payment methods for future transactions.
* The system should send payment confirmation notifications to the users via email or SMS.
* The system should integrate with third-party payment gateways to ensure secure transactions using SSL encryption.
* The system should support multiple languages, including English, French, and local languages in Africa and Asia.

# 5. Other Nonfunctional Requirements

* The system should support at least 10,000 concurrent users.
* The system should ensure video streams have a latency of less than 3 seconds.
* The system should prevent unauthorized access to user data or match streams.
* The system should use SSL for secure communication.
* The system should implement two-factor authentication for admin accounts.
* The system should be user-friendly with an intuitive interface.
* The system should have 99.9% uptime during live matches.
* The system should ensure that ads do not interrupt live streaming.
* The system should require users to agree to the terms of service before accessing streams.
* The system should comply with regional data protection laws (GDPR, CCPA).

# 9. Operation Contracts:

## 9.1 Contract UC1: LiveStreamMatch

* **Operation**: startLiveStream(matchID: MatchID)
* **Cross References**: Use Cases: Live Streaming of Soccer Matches.
* **Pre-conditions**: The user has selected a live match.
* **Post-conditions**:
  + A LiveStream instance was created. (instance creation)
  + The LiveStream was associated with the selected Match. (association formed)
  + The LiveStream quality was adjusted based on the user's network connection.
  + The LiveStream was associated with the User. (association formed)

## 9.2 Contract UC2: SubscribeToMatchUpdates

* **Operation**: subscribeToUpdates(matchID: MatchID)
* **Cross References**: Use Cases: Real-Time Match Updates.
* **Pre-conditions**: The user is logged in and a match is ongoing.
* **Post-conditions**:
  + A Subscription instance was created for the user. (instance creation)
  + The Subscription was associated with the selected Match. (association formed)
  + The system sends updates for each significant event in the match (e.g., goals, fouls) to the user.

## 9.3 Contract UC3: ManageUserAccount

* **Operation**: updateAccountDetails(userID: UserID, details: AccountDetails)
* **Cross References**: Use Cases: User Account Management.
* **Pre-conditions**: The user is logged in.
* **Post-conditions**:
  + The User's account information was updated with the new details.
  + The system confirmed the changes to the user. (confirmation)

## 9.4 Contract UC4: ManageContentByAdmin

* **Operation**: manageContent(contentID: ContentID, action: Action)
* **Cross References**: Use Cases: Admin Content Management.
* **Pre-conditions**: Admin is logged in with elevated privileges.
* **Post-conditions**:
  + The specified content instance Content was either approved, edited, or deleted based on the action.
  + The system recorded the moderation action for future reference. (association formed)
  + If content was flagged, the flag status was updated based on admin's action.

## 9.5 Contract UC5: AccessMatchReplayOrHighlights

* **Operation**: viewReplayOrHighlights(matchID: MatchID, type: ReplayType)
* **Cross References**: Use Cases: Match Replay and Highlights.
* **Pre-conditions**: The selected match has concluded.
* **Post-conditions**:
  + A ReplaySession or HighlightsSession instance was created. (instance creation)
  + The ReplaySession or HighlightsSession was associated with the selected Match. (association formed)
  + The session was associated with the User for tracking purposes. (association formed)

## 9.6 Contract UC6: JoinLiveChat

* **Operation**: joinLiveChat(matchID: MatchID)
* **Cross References**: Use Cases: Social Interaction and Commentary.
* **Pre-conditions**: The user is streaming or following a live match.
* **Post-conditions**:
  + A ChatSession instance was created. (instance creation)
  + The ChatSession was associated with the ongoing Match. (association formed)
  + The ChatSession was linked to the User. (association formed)
  + Messages sent by the user were displayed to other chat participants in real-time.

## 9.7 Contract UC7: ConfigureCustomNotifications

* **Operation**: setNotificationPreferences(userID: UserID, preferences: NotificationPreferences)
* **Cross References**: Use Cases: Custom Match Notifications.
* **Pre-conditions**: The user is logged in.
* **Post-conditions**:
  + The User's notification preferences were updated with the specified settings.
  + A NotificationProfile instance was associated with the User. (association formed)
  + The system tracks specific match events based on the user’s preferences. (association formed)

## 9.8 Contract UC8: UploadUserContent

* **Operation**: uploadContent(matchID: MatchID, content: ContentData)
* **Cross References**: Use Cases: User-Generated Match Content.
* **Pre-conditions**: The user is logged in and the match has concluded.
* **Post-conditions**:
  + A UserContent instance was created. (instance creation)
  + The UserContent was associated with the relevant Match. (association formed)
  + The UserContent was linked to the User for accountability. (association formed)
  + The system applied moderation filters to the UserContent to ensure compliance with platform guidelines.

## 9.9 Contract UC9: SubscribeToPremiumFeatures

* **Operation**: subscribeToPremium(userID: UserID, plan: SubscriptionPlan)
* **Cross References**: Use Cases: Premium Features for Users.
* **Pre-conditions**: The user is logged in.
* **Post-conditions**:
  + A PremiumSubscription instance was created for the user. (instance creation)
  + The PremiumSubscription was associated with the User. (association formed)
  + The User gained access to premium features such as ad-free streaming, multiple camera angles, and exclusive content.

## 9.10 Contract UC10: ProcessPayment

* **Operation**: makePayment(userID: UserID, amount: Currency, paymentMethod: PaymentMethod)
* **Cross References**: Use Cases: Payment Methods.
* **Pre-conditions**: The user selected a subscription plan or item to purchase.
* **Post-conditions**:
  + A Payment instance was created. (instance creation)
  + The Payment was associated with the User and the selected subscription or item. (association formed)
  + The payment status was updated to reflect success or failure.
  + The system sent a payment confirmation or failure notification to the user. (association formed)

# 15. GOF Design Patterns:

## 15.1 Creational Pattern: Singleton

* Ensures a single instance of the StreamingEngine to maintain consistency across streams.
* Used for the StreamingEngine to ensure a consistent and single streaming instance for all users. This avoids redundancy and synchronization issues.

Java Code Implementation:

public class StreamingEngine {

    private static StreamingEngine instance;

    // Private constructor to restrict instantiation

    private StreamingEngine() {

        System.out.println("Streaming Engine Initialized");

    }

    // Public method to provide the single instance

    public static synchronized StreamingEngine getInstance() {

        if (instance == null) {

            instance = new StreamingEngine();

        }

        return instance;

    }

    public void startStream(String matchName) {

        System.out.println("Streaming started for match: " + matchName);

    }

}

## 15.2 Structural Pattern: Adapter

* Adapts the VideoPlayer to handle multiple codecs (e.g., MP4, HLS).
* Used for VideoPlayer to handle compatibility with multiple video formats (MP4, HLS). This ensures seamless streaming regardless of the underlying codec.

Java Code Implementation

// Target Interface

public interface VideoPlayer {

void playVideo(String format);

}

// Adapter Class

public class CodecAdapter implements VideoPlayer {

private AdvancedVideoPlayer advancedPlayer;

public CodecAdapter(String format) {

advancedPlayer = new AdvancedVideoPlayer();

}

@Override

public void playVideo(String format) {

if (format.equalsIgnoreCase("MP4")) {

advancedPlayer.playMP4();

} else if (format.equalsIgnoreCase("HLS")) {

advancedPlayer.playHLS();

} else {

System.out.println("Unsupported format: " + format);

}

}

}

// Adaptee Class

public class AdvancedVideoPlayer {

public void playMP4() {

System.out.println("Playing video in MP4 format...");

}

public void playHLS() {

System.out.println("Playing video in HLS format...");

}

}

Example:

public class StreamingApplication {

public static void main(String[] args) {

VideoPlayer player = new CodecAdapter("MP4");

player.playVideo("MP4");

player = new CodecAdapter("HLS");

player.playVideo("HLS");

}

}

## 15.3 Behavioral Pattern: Observer

* Notifies users about match updates and breaking news in real-time.
* Used for NotificationCenter to manage real-time updates (e.g., goals, halftime, match completion). Observers (users) are notified dynamically when events occur.

Java Code Implementation:

import java.util.ArrayList;

import java.util.List;

// Observer Interface

interface Observer {

void update(String message);

}

// Concrete Observer

class User implements Observer {

private String userID;

public User(String userID) {

this.userID = userID;

}

@Override

public void update(String message) {

System.out.println("Notification for " + userID + ": " + message);

}

}

// Subject Interface

interface Subject {

void registerObserver(Observer observer);

void removeObserver(Observer observer);

void notifyObservers(String message);

}

// Concrete Subject

public class NotificationCenter implements Subject {

private List<Observer> observers = new ArrayList<>();

@Override

public void registerObserver(Observer observer) {

observers.add(observer);

}

@Override

public void removeObserver(Observer observer) {

observers.remove(observer);

}

@Override

public void notifyObservers(String message) {

for (Observer observer : observers) {

observer.update(message);

}

}

}

Example:

public class SoccerLiveNotifications {

public static void main(String[] args) {

NotificationCenter notificationCenter = new NotificationCenter();

User user1 = new User("User1");

User user2 = new User("User2");

// Register users

notificationCenter.registerObserver(user1);

notificationCenter.registerObserver(user2);

// Notify users of match events

notificationCenter.notifyObservers("Goal scored by Team A!");

notificationCenter.notifyObservers("Halftime: Team A 1 - 0 Team B");

}

}

# 16. GRASP Patterns

This section outlines how GRASP principles are applied in the design of the Soccer Live system. Each principle is described with its role and application in the system's architecture.

## 16.1 Creator

* Description: Assign responsibility for creating an object to the class that aggregates, uses, or contains the object.
* **Application:**
  + The Subscription Manager is responsible for creating user profiles and managing their data.
  + Reasoning: The Subscription Manager aggregates subscription and user data, making it the most suitable candidate to create user-related objects.

## 16.2 Information Expert

* Description: Assign responsibility to the class that has the necessary information to fulfill the responsibility.
* **Application:**
  + The Match Data Processor provides match statistics and real-time updates.
  + Reasoning: This class is already designed to handle match-related data and processes, aligning with the principle.

## 16.3 Controller

* Description: Assign responsibility for handling system events to a controller class that delegates the tasks to other classes.
* **Application:**
  + The Notification Controller manages all user-triggered notification events, such as goal alerts and match updates.
  + Reasoning: It acts as a central point for managing notification-related tasks.

## 16.4 Low Coupling

* Description: Reduce dependencies between classes to enhance modularity and flexibility.
* **Application:**
  + The Streaming Engine minimizes dependencies on third-party APIs for payment processing and codec handling.
  + Reasoning: This reduces the impact of changes in external services on core system functionality.

## 16.5 High Cohesion

* Description: Assign responsibilities so that classes have highly related tasks, ensuring clarity and maintainability.
* **Application:**
  + The LiveStream class is solely responsible for handling video streaming tasks.
  + Reasoning: This specialization prevents the class from being burdened with unrelated responsibilities.

## 16.6 Polymorphism

* Description: Design components to use a common interface for similar operations, improving flexibility and extensibility.
* **Application:**
  + Classes like ReplaySession, LiveStream, and HighlightsSession implement a shared interface for different types of content playback.
  + Reasoning: This approach makes it easier to add new content types in the future.

## 16.7 Pure Fabrication

* Description: Create a class solely for specific responsibilities, avoiding overloading existing classes.
* **Application:**
  + The Report Generator creates analytics and activity reports.
  + Reasoning: Isolating this functionality prevents clutter in other classes like Admin or Streaming Engine.

## 16.8 Indirection

* Description: Assign responsibility to an intermediate object to mediate between other components, reducing direct coupling.
* **Application:**
  + An API Gateway mediates between the Soccer Live platform and third-party services like streaming APIs and payment gateways.
  + Reasoning: It ensures secure and efficient communication without exposing internal details.

## 16.9 Design Improvements

* **Enhance User Personalization:**  
  The NotificationProfile could support more detailed preferences, such as notifications for specific players or match events, to align with requirements.
* **Extend Polymorphism:**  
  Introduce a base class or interface (e.g., StreamingSession) to unify LiveStream, ReplaySession, and HighlightsSession, improving extensibility.
* **Incorporate Moderation:**The Content and UserContent classes should be integrated with a ModerationService to handle inappropriate content automatically.
* **Strengthen Security:**  
  Add a dedicated SecurityManager class to handle two-factor authentication, as well as user and admin access control.
* **Integrate Analytics:**  
  Introduce an AnalyticsManager to analyze user behavior and stream performance, leveraging Report Generator.

# Appendix A: Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| API: | Application Programming Interface - a set of functions allowing the application to interact with external software. |
| Soccer Live: | The name of the platform designed to stream live soccer matches to users worldwide. |
| UI: | User Interface - the visual elements and screens that the user interacts with. |
| HTTP: | Hyper Text Transfer Protocol - the protocol used for communication between web browsers and servers. |
| HTTPS: | Hyper Text Transfer Protocol Secure - a secure version of HTTP that encrypts data exchanges. |
| SRS: | Software Requirements Specification - this document, detailing the functional and non-functional requirements of the system. |
| GDPR: | General Data Protection Regulation (EU) 2016/679, European Union, 2016 - EU law on data protection and privacy. |
| CCPA: | California Consumer Privacy Act, USA, 2018 - a state statute intended to enhance privacy rights and consumer protection. |
| Adaptive Streaming: | A technique to deliver video streams by adjusting quality based on the user's network speed. |

# Appendix B: Analysis Models

This section is for including any diagrams and models that provide a detailed understanding of the system. For **Soccer Live**, you might include:

* **Class Diagrams**:  
  A class diagram illustrating the software architecture, including classes for users, streams, and matches.
* **Data Flow Diagram (DFD)**:  
  A DFD showing how data moves between user interfaces, streaming servers, and databases.
* **Entity-Relationship Diagram (ERD)**:  
  An ERD illustrating the relationships between users, matches, streams, and event notifications in the database.
* **Use Case Diagrams**:  
  Diagrams showing how users interact with the system for actions like streaming, receiving updates, and managing their accounts.

# Appendix C: To Be Determined List

This section tracks anything that has not been fully defined yet and needs further clarification or decision. For Soccer Live, some possible TBD items could include:

|  |  |
| --- | --- |
| **TBD Item** | **Description** |
| Streaming Protocol Decision | The choice between using HLS or DASH for adaptive video streaming (Pending research). |
| Third-Party API Selection | The decision regarding which third-party API provider to use for match data (e.g., Opta, Sportradar). |
| Ad Integration | Finalizing the plan for integrating advertisements in a way that doesn’t interfere with user experience. |
| Scalability Testing Framework | The framework to be used for load testing the system under high traffic conditions (Pending evaluation). |
| Localization Languages | Deciding on the additional languages to support beyond English (Pending market analysis). |