STM JOURNALS

Journal of
Artificial Intelligence
Research & Advances

ISSN: 2395-6720
Volume 11, Issue 2, 2024
May–August
DOI Journal: 10.37591/JoAIRA

https://journals.stmjournals.com/joaira

Research JoAIRA

# GPT in Code Conversion: Achieving Agile, Accurate, and Effective Translations Across Programming Languages

Prashant D. Sawant*

***Abstract***

*This comprehensive study meticulously investigates the intricacies and challenges inherent in the process of code conversion across a diverse range of programming languages. It provides an in-depth examination of select instances where code conversion poses significant challenges, and thoughtfully proposes innovative solutions to these complex problems. A key highlight of this research is the use of Generative Pre-trained Transformers (GPTs) like Copilot, which have proven to be instrumental in achieving fast, accurate, and effective code conversions. The study demonstrates how GPTs have solved difficult code conversions with a few intricate examples. The research further delves into the broader implications of these solutions, assessing their impact on the field of programming and software development. Additionally, the study identifies and discusses potential areas for future research, paving the way for further advancements in this critical aspect of programming. This research serves as a valuable resource for both novice and experienced developers navigating the complexities of code conversion.*

**Keywords:** Generative Pre-trained Transformers (GPTs), code conversion, software development, programming languages, copilot, artificial intelligence

## INTRODUCTION

The advent of artificial intelligence (AI), particularly Generative Pre-trained Transformers (GPTs), has ignited the spark of the 4th Industrial Revolution [1–3]. This revolution is characterized by the fusion of the physical, digital, and biological worlds, profoundly transforming all disciplines, economies, and industries.

GPTs have revolutionized the value chain processes, making them more accessible and streamlined [3–5]. They have introduced a new level of agility and automation, transforming the way we interact with technology [1–5]. However, this transformation is not without its challenges. One such challenge is the phenomenon of "hallucinations," where GPTs generate outputs that are plausible-sounding but factually incorrect [5].

***Author for Correspondence**
Prashant D. Sawant
E-mail: prasdsaw@gmail.com

A.I. Research Director, AI-D Consultancy, Melbourne, Australia

Received Date: April 23, 2024
Accepted Date: April 30, 2024
Published Date: June 20, 2024

**Citation:** Prashant D. Sawant. GPT in Code Conversion: Achieving Agile, Accurate, and Effective Translations Across Programming Languages. Journal of Artificial Intelligence Research & Advances. 2024; 11(2): 11–20p.

In the realm of software development, one of the critical challenges is the efficient and effective conversion of code [6, 7]. GPTs have shown immense potential in addressing this issue, accelerating the application creation process and allowing coding specialists to focus on higher-level, creative, more complex activities [5].

Code conversion, the process of translating code from one programming language to another, is a prevalent practice in software development [8]. It is often necessitated by factors such as migrating

© STM Journals 2024. All Rights Reserved 

software to a new platform, integrating systems that use different languages, or leveraging the strengths of a different language for performance, scalability, or the availability of certain libraries or frameworks. However, the process is not straightforward due to the unique syntax, semantics, and idioms of each programming language [9, 10]. Not all concepts or libraries in one language have a direct equivalent in another, leading to potential discrepancies in the behavior of the converted code [9–11]. Therefore, thorough testing of the converted code is imperative to ensure its functionality aligns with the original. In the context of specific tasks such as financial data analysis, market research data fetching, or HR management data reading, code conversion can facilitate the integration of code with systems that use different languages. For instance, one might need to convert Python code to Java for a large-scale financial system built with Java, or JavaScript code to Ruby for a web application developed with Ruby on Rails. Real-world code conversion involves more complex code and requires a comprehensive understanding of both the source and target languages. In some cases, it might be more efficient to rewrite the code from scratch in the target language, particularly if the original code is not well-structured or if the target language offers a more efficient solution. Therefore, the decision to convert code should be made judiciously, considering the complexity of the code, the differences between the source and target languages, and the specific project requirements [12–14].

**Code conversion/migration Challenges**

Some of these challenges are discussed as follows [8, 9]:

- *Dynamic vs. Static Typing:* Some languages like Python are dynamically typed, meaning the type is checked at runtime. On the other hand, languages like Java are statically typed, where the type is checked at compile time. Converting code between these types of languages can be challenging.
- *Memory Management:* Languages like C++ allow direct manipulation of memory through pointers, while others like Java manage memory automatically. Converting code that manually manages memory to a language that does not support this can be difficult.
- *Functional vs. Object-Oriented Programming:* Languages like Haskell are functional programming languages, while others like Java are object-oriented. Converting code between these paradigms requires a change in how the code is structured and thought about.
- *Error Handling:* Different languages have different methods of error handling. For example, Python uses exceptions, Go uses multiple return values, and Rust uses a type system. Converting the error handling code between these languages can be complex.
- *Concurrency Model:* Languages have different models for handling concurrency. For example, JavaScript has an event-driven, non-blocking I/O model, while Java uses a multi-threading model. Converting code between these models can be challenging.
- *Language-Specific Constructs:* Some languages have unique features that do not have a direct equivalent to other languages. For example, Python's list comprehensions or decorators, or JavaScript's prototype-based inheritance.

These challenges highlight the importance of having a deep understanding of both the source and target languages when performing code conversion. There are several non-GPT tools and best practices that can help ensure a smoother transition of complex code migration from one programming language or platform to another are as follows:

**Tools Used in Code Migration [15, 16]**

- *Automated Code Conversion Tools:* Tools like Mobilize.Net, AWS App2Container, Google Cloud Migrate for Anthos, and IBM Cloud Transformation Advisor can automate some aspects of the code conversion process.
- *Integrated Development Environments (IDEs):* IDEs like Eclipse, IntelliJ IDEA, and Visual Studio provide features that can help with code migration, such as code refactoring tools, syntax highlighting, and error detection.
- *Version Control Systems:* Tools like Git, Mercurial, and Subversion can help manage different versions of the codebase during the migration process.

© STM Journals 2024. All Rights Reserved 

- *Continuous Integration/Continuous Deployment (CI/CD) Tools:* Tools like Jenkins, Travis CI, and CircleCI can automate the building, testing, and deployment of the migrated code, ensuring that any issues are caught and addressed promptly.

Some of the best practices in code conversion or migration are as follows:

**Best Practices in Code Conversion/Migration [16]**

- *Begin with the End in Mind:* Understanding target architecture and how it benefits the project before starting the migration.
- *Inventory Time:* Taking stock of the code, determine what components are essential and what can be refactored.
- *Choosing Code Migration Strategy:* Decide if we want to complete migration in one go or a staged approach, upgrading parts of the application over time.
- *Testing:* Thoroughly test the application in its new environment. Unit tests, integration tests, and user acceptance testing are crucial here.
- *Monitoring Application:* Once everything is moved in and working as expected, monitor the application closely for any issues that might arise and be ready to make quick fixes.

The migration is not just about moving things from point A to point B, but it can help to declutter, find truly valuable codes, and enhance them that were perhaps not possible before [16]. Recently, articles have been emerging using GPT for code conversions/migrations [17, 18]. However, it is important to note that while GPTs can assist with code conversion, they are not perfect. The generated code should always be reviewed and tested by a human developer to ensure it behaves as expected. Also, GPTs currently do not have the ability to understand the semantics of the code at a deep level, so they might not always be able to handle complex code conversion tasks that require a deep understanding of the source and target languages. But they are continually improving and can already provide significant assistance in many code conversion tasks.

This study explores the pros and cons associated with code conversion, focusing on cases where it is particularly difficult to convert code from one language to another and how GPTs like Copilot can help in efficient code conversion.

## METHODS AND MATERIALS

The study uses a comparative approach, examining code snippets in different programming languages and attempting to convert them. The languages chosen for this study are Java, Python, and C++, due to their widespread use and differing characteristics. The code snippets are selected based on their complexity and the unique features of their original language that may pose challenges in conversion.

## STUDY DESIGNS AND CASES

The study is divided into several cases, each focusing on a specific code snippet and the challenges and solutions in converting it to another language.

## RESULTS AND DISCUSSION

In the realm of software development, particularly in a multi-language environment, developers often encounter challenges when transitioning between different programming languages. The subsequent quintet of instances elucidates the process of proficient code conversion, while concurrently delineating the potential challenges inherent in such tasks. These examples not only demonstrate the conversion process but also provide a comprehensive understanding of the complexities involved in code conversion. This exploration aims to shed light on the intricacies of code conversion, thereby fostering a deeper understanding of this critical aspect of software engineering.

© STM Journals 2024. All Rights Reserved 

**Example 1: Python List Comprehensions and Equivalent JavaScript Code**

*Problem Statement:* The translation of Python list comprehension to equivalent JavaScript code presents a significant challenge for developers [19, 20]. Python list comprehension provides a concise and readable way to create and manipulate lists. However, developers transitioning from Python to JavaScript may find it difficult to perform equivalent operations due to the gap between Python's list comprehension and JavaScript's map() function. This issue necessitates a solution that can bridge this gap, particularly for developers working in a multi-language environment.

Python list comprehension provides a concise way to create lists based on existing lists (Figure 1).

In JavaScript, we can achieve a similar result using the map() function (Figure 2).

*Problem Solved:* The solution involves a code conversion that translates Python list comprehension to equivalent JavaScript code. In the given Python code, a list of numbers is squared using list comprehension. The equivalent operation in JavaScript is performed using the map() function, which applies a function to each item in an array and returns a new array with the results. This code conversion bridges the gap between Python and JavaScript, making it easier for developers to understand and write code in both languages.

**Example 2: C++ Pointers and Equivalent Java Code**

*Problem Statement:* The challenge lies in translating the concept of pointers from C++ to the equivalent concept in Java. C++ uses pointers to store the memory address of a variable and accesses the value at that memory address using the dereference operator (*) [21, 22]. However, Java uses a system of references, which automatically handles the referencing and dereferencing of objects. This difference in memory management can pose a significant hurdle for developers transitioning from C++ to Java. The need is for a solution that bridges this gap, making it easier for developers to understand and write code in both languages.

In C++, pointers are used to store the memory address of a variable (Figure 3).

Figure 4 depicts the equivalent Java Code.

```python
# Python code
numbers = [1, 2, 3, 4, 5]
squares = [n**2 for n in
numbers]
print(squares)  # Output: [1, 4,
9, 16, 25]
```

**Figure 1.** Python code for list comprehension.

```javascript
// JavaScript code
let numbers = [1, 2, 3, 4, 5];
let squares = numbers.map(n =>
n**2);
console.log(squares);  //
Output: [1, 4, 9, 16, 25]
```

**Figure 2.** Equivalent JavaScript Code created using CoPilot.

© STM Journals 2024. All Rights Reserved 

```cpp
// C++ code
int number = 5;
int* p = &number;
cout << *p;  // Output: 5
```

**Figure 3.** C++ Pointers.

```java
// Java code
int number = 5;
Integer numReference = number;
System.out.println(numReference)
;  // Output: 5
```

**Figure 4.** Equivalent Java Code using CoPilot.

```python
import pandas as pd
import pandas_datareader as
pdr
data =
pdr.get_data_yahoo('AAPL',
'2020-01-01', '2020-12-31')
close = data['Close']
returns = close.pct_change()
print(returns)
```

**Figure 5.** Python Code (Pandas for financial data analysis).

*Problem Solved:* The solution involves a code conversion that translates the concept of pointers from C++ to the equivalent concept in Java. In the equivalent Java code, an Integer object is used to hold the value of the number variable, which can be accessed directly without needing to dereference a pointer. This conversion bridges the gap between C++ and Java, and highlights the differences between the two languages, such as C++'s manual memory management with pointers vs. Java's automatic memory management with garbage collection and references.

**Example 3: Python to Java (Financial Data Analysis)**

*Problem Statement:* In the field of financial data analysis, integrating Python-based analysis, specifically using the Pandas library, into a larger Java-based system presents a significant challenge [23, 24]. The interoperability issues between Python and Java can lead to difficulties in integration and affect code maintainability. The need, therefore, is for a solution that enables equivalent financial data analysis in a Java-based system, ensuring seamless integration and improved maintainability.

Python Code (Pandas for financial data analysis) has been shown in Figure 5.

The equivalent Java Code Using the TA-Lib library for financial data analysis) created using Copilot has been shown in Figure 6.

*Problem Solved:* The solution involved converting the original Python code, which used the Pandas library for financial data analysis, to Java. By using the TA-Lib library in the converted Java code, we were able to perform the same financial data analysis while maintaining consistency with the rest of the Java-based system, ensuring seamless integration and improved code maintainability.

© STM Journals 2024. All Rights Reserved 

```
import
com.tictactec.ta.lib.*;
double input[] = { /* Your
data here */ };
MInteger outBegIdx = new
MInteger();
MInteger outNBElement = new
MInteger();
double output[] = new
double[input.length];
RetCode retCode = Core.roc(0,
input.length-1, input, 10,
outBegIdx, outNBElement,
output);
if (retCode ==
RetCode.Success) {
    for (int i = 0; i <
outNBElement.value; i++) {

System.out.println(output[i])
;
    }
}
```

**Figure 6.** The equivalent Java Code (with the TA-Lib library) created using Copilot.

```
const axios =
require('axios');
axios.get('https://api.exampl
e.com/market-research-data')
    .then(response => {

console.log(response.data);
    })
    .catch(error => {
        console.error(error);
    });
```

**Figure 7.** JavaScript Code.

**Example 4: JavaScript to Ruby (Fetching Market Research Data)**

*Problem Statement:* In the realm of web application development, especially with Ruby on Rails, integrating JavaScript code that fetches market research data from an API presents a significant challenge [25, 26]. The use of multiple languages can complicate code management, maintenance, and debugging. The need, therefore, is for a solution that implements the data fetching functionality in Ruby, ensuring seamless integration with the Ruby on Rails framework and improved code maintainability.

JavaScript Code (Fetching market research data from an API has been shown in Figure 7.

© STM Journals 2024. All Rights Reserved 

```ruby
require 'net/http'
require 'json'
uri =
URI('https://api.example.com/
market-research-data')
response = Net::HTTP.get(uri)
data = JSON.parse(response)
puts data
```

**Figure 8.** Ruby Code.

Ruby Code (Fetching market research data from an API) created using Copilot has been shown in Figure 8.

*Problem Solved:* The solution involved converting the original JavaScript code, which fetched market research data from an API to Ruby. This conversion allowed the development team to manage all code in a single language, simplifying maintenance and debugging. Furthermore, it enhanced integration with the Ruby on Rails framework, leading to a more efficient web application development process.

**Example 5: Java to Python (Reading HR Management Data)**

*Problem Statement:* In HR management data analysis, the use of Java for reading data from a CSV file presents a challenge. Despite Java's robust capabilities, it lacks specialized libraries like Pandas in Python for efficient data analysis tasks. This leads to complex and less efficient data preprocessing steps. The need, therefore, is for a solution that implements the same data reading functionality in Python, leveraging its powerful data analysis capabilities for more efficient and readable code.

In the context of data analysis, particularly in HR management, a significant challenge arises when using Java to read data from a CSV file. While Java is a powerful language, it may not be the most efficient for data analysis tasks due to the lack of specialized libraries like Pandas in Python [27, 28]. This discrepancy can lead to complex and less efficient data preprocessing steps. Therefore, there is a need for a solution that allows the same data reading functionality to be implemented in Python, ensuring more efficient and readable code, and leveraging Python's powerful data analysis capabilities.

Java Code (Reading HR management data from a CSV file) has been shown in Figure 9.

Equivalent Python Code (Reading HR management data from a CSV file) is produced by Copilot as shown in Figure 10.

*Problem Solved:* The solution involved converting the original Java code, which read HR management data from a CSV file to Python. This change was particularly beneficial for a team performing data analysis tasks, as Python, with its powerful data analysis libraries like Pandas, is a more suitable language. The conversion simplified the data preprocessing steps, leading to more efficient and readable code.

In all above-mentioned examples, code conversion helped solve specific problems related to system integration, code maintainability, and the efficient use of language-specific libraries. It is important to note that the decision to convert code should be based on the specific needs and constraints of the project, and the converted code should always be thoroughly tested to ensure it behaves as expected. These examples also illustrate some of the challenges in converting code between different programming languages. Each language has its own unique features and syntax, and a feature in one language may not have a direct equivalent in another. This is why code conversion requires a deep understanding of both the source and target languages.

© STM Journals 2024. All Rights Reserved 

```
import
java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class Main {
    public static void
main(String[] args) {
        try (BufferedReader
br = new BufferedReader(new
FileReader("hr_data.csv"))) {
            String line;
            while ((line =
br.readLine()) != null) {

System.out.println(line);
            }
        } catch (IOException
e) {

e.printStackTrace();
        }
    }
}
```

**Figure 9.** Java Code HR management data from a CSV file.

```
import pandas as pd
df =
pd.read_csv('hr_data.csv')
print(df)
```

**Figure 10.** Equivalent Python Code produced by Copilot.

## PROS AND CONS USING GPTS FOR CODE CONVERSION

The pros and cons of using Generative Pretrained Transformers (GPTs) like Copilot for code conversion are as follows:

### Pros

- *Language Agnostic:* GPTs are trained on a diverse range of internet text, so they can generate code in any programming language.
- *Efficiency:* GPTs can quickly generate code, saving developers time and effort.
- *Learning Tool:* GPTs can serve as a learning tool for developers who are new to a language, providing examples of how to write certain functions or structures.
- *Agility for Software Development Life Cycle (SDLC):* The code conversion can allow software to leverage the strengths of a different language, or to be used on a new platform, bringing agility for SDLC.

### Cons

- *Accuracy:* While GPTs are powerful, they can sometimes generate incorrect or inefficient code, leading to potential errors.
- *Loss of Original Language Features:* During the code conversion, there is potential for the loss of original language features which might not have a direct equivalent in the target language.

© STM Journals 2024. All Rights Reserved 

- *Context-Awareness:* GPTs might not fully understand the context or the semantics of the code, which can lead to generation of code that is syntactically correct but semantically incorrect.
- *Security:* There could be potential security risks if sensitive information is included in the code for conversion.

While GPTs can be a powerful tool for code conversion, it is important for developers to review and understand the generated code to ensure its correctness and efficiency. It is also crucial to consider the potential loss of original language features and to manage any potential security risks.

## CONCLUSION

The study concludes that code conversion, despite its inherent complexities and challenges, presents a fertile ground for further exploration and advancement. The use of sophisticated tools and techniques, particularly Generative Pre-trained Transformers like Copilot, has the potential to revolutionize the process of code conversion. By enhancing accuracy and efficiency, these advancements promise significant benefits for the broader software development industry, paving the way for more seamless and effective multi-language programming environments.

## FUTURE WORK

The future work could focus on several promising areas.

- Firstly, while GPTs like Copilot have proven effective in code conversion, there is room to explore other AI models and techniques that could further enhance the efficiency and accuracy of code conversion.
- Secondly, given the potential security risks identified, future research could focus on developing robust security measures to ensure sensitive information is adequately protected during the code conversion process.
- Thirdly, as GPTs might not fully understand the context or semantics of the code, research could be directed towards improving the context-awareness of these models to generate code that is not only syntactically correct but also semantically accurate.
- Lastly, considering the potential loss of original language features during code conversion, future studies could investigate methods to preserve these features or provide equivalent functionality in the target language.

By addressing these areas, future research can contribute to the development of more advanced, secure, and context-aware code conversion tools, further revolutionizing the software development industry and enhancing multi-language programming environments.

## REFERENCES

1. Prashant Sawant. Artificial Intelligence: The Era of New Industrial Revolution. Kindle Edition. 2022.
2. Bristol H, Enno de Boer, Dinu de Kroon, Shahani R, Torti F. (2024 Feb 21). Adopting AI at speed and scale: The 4IR push to stay competitive. [Online]. McKinsey & Company. Available from: https://www.mckinsey.com/capabilities/operations/our-insights/adopting-ai-at-speed-and-scale-the-4ir-push-to-stay-competitive
3. Crafts N. Artificial intelligence as a general-purpose technology: an historical perspective. Oxford Rev Econ Policy. 2021 Sep 1; 37(3): 521–536.
4. Davio Larnout. (2023 Apr 18). The Evolution of GPTs: Current State and Future Predictions. [Online]. Radix.ai. Available from: https://insights.radix.ai/blog/the-evolution-of-gpts-current-state-and-future-predictions
5. Prashant D. Sawant. A Real-Time Visualization Framework to Enhance Prompt Accuracy and Result Outcomes Based on Number of Tokens. J Artif Intell Res Adv. 2024; 11(1): 44–52.
6. BCG Platinion. (2023 Apr 26). GPT's Impact on the Software Development Lifecycle. [Online]. Available from: https://bcgplatinion.com/insights/gpt-impact-on-sdlc/

© STM Journals 2024. All Rights Reserved 

7. KPMG. (2023). How generative AI can revolutionize the software development lifecycle. [Online]. Available from: https://kpmg.com/us/en/articles/2023/generative-artificial-intelligence-software-development-lifecycle.html
8. McGowan J, Weary R, Carriere L, Game ET, Smith JL, Garvey M, Possingham HP. Prioritizing debt conversion opportunities for marine conservation. Conserv Biol. 2020 Oct; 34(5): 1065–75.
9. Liu C, Xia X, Lo D, Gao C, Yang X, Grundy J. Opportunities and challenges in code search tools. ACM Comput Surv (CSUR). 2021 Oct 7; 54(9): 1–40.
10. Kim Y, Lee M. A Comparative Study of Educational Programming Languages for Non-majors Students: from the Viewpoint of Programming Language Design Principles. The Journal of Korean Association of Computer Education. 2019; 22(1): 47–61.
11. Borji A. A categorical archive of chatgpt failures. arXiv preprint arXiv:2302.03494. 2023 Feb 6.
12. Chinmay Kappor. (2024 Jan 10). Harnessing the Power of Large Language Models for Efficient Code Conversion. [Online]. Medium. Available from: https://medium.com/@kapoorchinmay231/large-language-models-llms-for-code-conversion-new-age-of-ai-72ebd2c8918d
13. McKay S. (2023). Code Language Translator: Translate Code Between Languages | Master Data Skills + AI. Insights and Strategies from the Enterprise. [Online]. DNA Blog. Available from: https://blog.enterprisedna.co/code-language-translator/
14. Kajol Aikat. (2021 Sep 27). 5 Best tools to convert cross platform codes in 2021. [Online]. TechGig. Available from: https://content.techgig.com/technology/5-best-tools-to-convert-cross-platform-codes-in-2021/articleshow/86257816.cms
15. Luke Shannon. (2022 Aug 26). Modernization: Developing your code migration strategy. [Online]. Redhat.com. Available from: https://www.redhat.com/en/blog/modernization-developing-your-code-migration-strategy
16. Walsh D. (2024 Feb 18). Best Practices for a Successful Code Migration. [Online]. Mobilize.net. GAP. Available from: https://www.mobilize.net/blog/best-practices-for-a-successful-code-migration
17. Zhang Z, Zhang L, Yuan X, Zhang A, Xu M, Qian F. A First Look at GPT Apps: Landscape and Vulnerability. arXiv preprint arXiv:2402.15105. 2024 Feb 23.
18. Zheng Y. ChatGPT for teaching and learning: An experience from data science education. In Proceedings of the 24th Annual Conference on Information Technology Education. 2023 Oct 11; 66–72.
19. Python. (2024). Data Structures 5. Python documentation. [Online]. Available from: https://docs.python.org/3/tutorial/datastructures.html
20. Developer: Mozilla. (2023). Array.prototype.map() - JavaScript: MDN. [Online]. MDN Web Docs. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map
21. Stroustrup B. The C++ Programming Language. 4th Edn. Addison-Wesley Professional; Boston. 2013.
22. Gosling J, Joy B, Steele G, Bracha G, Buckley A. The Java Language Specification. Java SE 8th Edn. Addison-Wesley Professional; Boston. 2014.
23. McKinney W. Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference. 2010; 51–56.
24. Paul N, Evans D. Comparing Java and. NET security: Lessons learned and missed. Comput Secur. 2006 Jul 1; 25(5): 338–50.
25. Luecke KR, Ellis BJ, Baxter I, Akers RL, Mehlich M. Software code base conversions. In2007 IEEE/AIAA 26th Digital Avionics Systems Conference. 2007 Oct 21; 6-C.
26. Flanagan D. JavaScript: The Definitive Guide. O'Reilly Media; California, United States. 2006.
27. McKinney W. Data structures for statistical computing in Python. InSciPy. 2010 Jun 28; 445(1): 51–56.
28. Arnold K, Gosling J, Holmes D. The Java Programming Language. 4th Edn. Addison-Wesley Professional; Boston. 2005.

© STM Journals 2024. All Rights Reserved 