Bridging the Gap between Source Code and Requirements Using GPT
(Student Abstract)

Ruoyu Xu, Zhengyu Xu, Gaoxiang Li, Victor S. Sheng
Computer Science Department, Texas Tech University, Lubbock, Texas, USA
{ruoyxu, zhenxu, gaoli, victor.sheng}@ttu.edu

Abstract
Reverse engineering involves analyzing the design, architecture, and functionality of systems, and is crucial for legacy systems. Legacy systems are outdated software systems that are still in use and often lack proper documentation, which makes their maintenance and evolution challenging. To address this, we introduce SC2Req, utilizing the Generative Pre-trained Transformer (GPT) for automated code analysis and requirement generation. This approach aims to convert source code into understandable requirements and bridge the gap between those two. Through experiments on diverse software projects, SC2Req shows the potential to enhance the accuracy and efficiency of the translation process. This approach not only facilitates faster software development and easier maintenance of legacy systems but also lays a strong foundation for future research, promoting better understanding and communication in software development.

Introduction
The rapid growth of digital technology has led to software systems becoming an indispensable part of businesses and organizations across various industries. However, many software systems have evolved over decades and are now referred to as legacy systems (Warren 2012). These legacy systems are generally large, complex, and poorly documented. Understanding the architecture of these systems is essential for managing them. Reverse engineering helps developers understand operations, identify flaws, and improve systems. Applications of this process include malware detection, data recovery, and maintenance and improvement of legacy systems (García-Borgeñoñon et al. 2023).

Building on this, researchers have implemented various tools for reverse engineering and automating source code documentation. Technologies such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which are applied to generate summaries based on code patterns and structures (Iyer et al. 2016). Khan and Uddin utilized OpenAI’s Codex to automatically generate documentation for the code (Khan and Uddin 2022). Ahmad et al. harnessed Transformer models for automated documentation and summarization of source code (Ahmad et al. 2020). Although some studies have concentrated on documenting specific code units, such as methods and functions, they lack a comprehensive solution for obtaining the requirements necessary for software development or maintenance.

In this study, we introduce SC2Req, which uses the Generative Pre-trained Transformer (GPT) to convert source code into requirements. Combining GPT’s code analysis strength with natural language generation, SC2Req effectively bridges the code-requirement gap. Tested across various software projects, SC2Req demonstrated its potential in aligning documentation with true system requirements, offering a streamlined approach to software development and legacy system maintenance.

Experiment
Datasets and Preprocessing
We collected source code and requirement pairs from six software systems spanning different domains. These include Albergate (55 requirements), SMOS (1073 requirements), EBT (98 requirements), eTour (310 requirements), iTrust (534 requirements), and eAnci (567 requirements). In our six-fold cross-validation experiment, each iteration held out one dataset as an unseen test set, using the rest five datasets for training (80%) and validation (20%).

Although we intended to evaluate various programming languages, obtaining varied real-world paired datasets proved challenging. Therefore, we utilized the GPT-3 text-DaVinci-003 model to convert Java source code from our projects into Python and JavaScript, generating multi-language datasets. For correctness, we employed tests and expert reviews. Italian datasets like Albergate and eAnci were translated to English using DeepL and verified by native Italian computer science students. After data cleaning, we had a final count of 2,637 code and requirement pairs.

For SC2Req’s assessment, we benchmarked against the original GPT-3 model (i.e., Basic GPT) and a human-based approach (i.e., Human Rewritten). For the Human Rewritten method, we engaged 20 computer science graduate students, training them to rephrase requirements from source codes.
performs Basic GPT in all programming languages. As Ta-

data (show in Table 2). Notably, SC2Req consistently out-
claims 1 and 2.

tables 1 and 2.

Score and Semantic Similarity metrics, as presented in Ta-

ing languages (i.e., Java, Python, JavaScript) using BLEU

Rewritten) were evaluated on six datasets (i.e., Albergate,

Three methods (i.e., SC2Req, Basic GPT, and Human

Results and Analysis

ute challenges for maintenance, and understanding their re-

Process of semantics and structure but also curtails manual

put. This automation not only streamlines the understanding

of the original text, thus delivering a truer representation of

methods. Specifically, it recreates a more substantial portion

of the original text, thus delivering a truer representation of

that are often overlooked by benchmark methods, further un-

understanding its qualitative superiority.

References

Ahmad, W. U.; Chakraborty, S.; Ray, B.; and Chang, K.-W. 2020. A transformer-based approach for source code sum-


Black, S.; Gao, L.; Wang, P.; Leahy, C.; and Biderman, S. 2022. Gpt-neo: Large scale autoregressive language model-


Khan, J. Y.; and Uddin, G. 2022. Automatic Code Documenta-


SMOS 35.39 0.81

eTour 29.11 0.66
eAnci 34.55 0.77

Table 2: Experimental results when each dataset is held out

Albergate 33.05 0.72

Table 1: Experimental results when each dataset attends

Albergate SC2Req 38.22 0.74

Basic GPT 18.74 0.58

Human Rewritten 36.33 0.73

SMOS SC2Req 45.73 0.86

Basic GPT 36.33 0.73

Human Rewritten 47.38 0.88

EBT SC2Req 48.85 0.90

Basic GPT 20.36 0.56

Human Rewritten 49.80 0.92

eTour SC2Req 44.63 0.83

Basic GPT 25.66 0.60

Human Rewritten 45.37 0.85

iTrust SC2Req 37.21 0.71

Basic GPT 21.95 0.57

Human Rewritten 36.05 0.71

eAnci SC2Req 42.55 0.81

Basic GPT 28.12 0.62

Human Rewritten 40.62 0.77

Table 1: Experimental results when each dataset attends training process in Java programming language.

Automating Code-to-Requirements Translation

with GPT-Neo

Complex software and under-documented legacy systems pose challenges for maintenance, and understanding their require-

ments is key. We introduce an automated method that uses the GPT model to transform source code into require-

ments, bridging the code-documentation gap.

GPT’s prowess in natural language processing tasks in-

spired its adaptation for the code-to-requirements translation

endeavor. We chose the GPT-Neo 2.7B (Black et al. 2022), a more accessible and cost-effective alternative to GPT-3, with 2.7 billion parameters compared to GPT-3’s 175 bil-

lion. We fine-tuned it on our specific code-to-requirements

datasets. Undertaking 1,000 training steps using zero-shot

training process for SC2Req in Java programming language

ble 1 illustrates, SC2Req achieves a BLEU score of 38.22

and semantic similarity of 0.74 on the Albergate dataset in

Java, while Basic GPT achieves scores of 18.74 and 0.58 re-

spectively. SC2Req’s performance is comparable to that of

the Human Rewritten method. In Table 1, SC2Req achieves

scores of 44.63 in BLEU and 0.83 in semantic similarity on

the eTour dataset, while Human Rewritten scores are slightly

higher at 45.37 and 0.85 respectively. In the hold-out ex-

periment (Table 2), SC2Req maintains commendable perfor-

mance across various languages. In Java, the BLEU scores

ranged from 27.07 (eTour) to 42.98 (EBT).

In the qualitative analysis, SC2Req presents distinct ad-

vantages over both the Basic GPT and Human Rewritten

methods. Specifically, it recreates a more substantial portion

of the original text, thus delivering a truer representation of

the content. Moreover, SC2Req avoids adding extraneous el-

ements, which ensures clarity in understanding the depicted

processes and requirements. It also prioritizes vital aspects

that are often overlooked by benchmark methods, further un-

derscoring its qualitative superiority.

Results and Analysis

Three methods (i.e., SC2Req, Basic GPT, and Human

Rewritten) were evaluated on six datasets (i.e., Albergate,

SMOS, EBT, eTour, iTrust, and eAnci) in three program-

ming languages (i.e., Java, Python, JavaScript) using BLEU

Score and Semantic Similarity metrics, as presented in Ta-

bles 1 and 2.

SC2Req’s performance is analyzed in two scenarios:

without hold-out data (show in Table 1) and with hold-out

data (show in Table 2). Notably, SC2Req consistently out-

performs Basic GPT in all programming languages. As Ta-


The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)