Chandra Credit Software License (CCSL)

Technical Documentation and Implementation Guide

Version 0.1.0

Shyamal Chandra

May 5, 2025

Contents

Chapter 1

Introduction

1.1 Overview

The Chandra Credit Software License (CCSL) is a revolutionary approach to software licensing that integrates code quality metrics with cryptocurrency micropayments. Unlike traditional licensing models that focus solely on usage rights, CCSL establishes a direct relationship between code quality, usage, and compensation.

1.2 Motivation

Software development is increasingly collaborative, yet current licensing models fail to adequately recognize and reward individual contributions based on their quality. The CCSL system aims to solve this problem by:

- Objectively measuring code quality through defined metrics
- Enabling fair compensation proportional to contribution value
- Creating a more sustainable open-source ecosystem
- Leveraging cryptocurrency for efficient micropayments

1.3 Key Components

The CCSL system consists of three main components:

1. **License Management**: Tracks code contributions and their associated metadata
2. **Metrics Evaluation**: Evaluates code quality using six distinct metrics
3. **Payment Integration**: Facilitates Bitcoin micropayments to contributors

Chapter 2

License Management

2.1 Core Classes

The license management component provides classes for tracking code contributions and managing license information.

2.1.1 License Class

The `License` class serves as the central point for managing a CCSL-licensed project:

```
class License {
public:
    License(const std::string& projectName, const std::string& licenseKey);

    bool registerContribution(const CodeContribution& contribution);
    bool validate() const;
    std::string getLicenseInfo() const;

    // Getters
    const std::string& getProjectName() const;
    const std::string& getLicenseKey() const;
    PaymentManager& getPaymentManager();
    const std::vector<CodeContribution>& getContributions() const;

private:
    std::string m_projectName;
    std::string m_licenseKey;
    std::vector<CodeContribution> m_contributions;
    PaymentManager m_paymentManager;
};
```

2.1.2 CodeContribution Class

The `CodeContribution` class represents a specific contribution to the codebase:

```
class CodeContribution {
public:
    CodeContribution(const std::string& contributor,
                     const std::string& fileId,
                     int lineStart,
                     int lineEnd);

    void addMetricEvaluation(const MetricEvaluation& evaluation);
    double calculateValue() const;

    // Getters
    const std::string& getContributor() const;
    const std::string& getFileId() const;
    std::pair<int, int> getLineRange() const;
```

```
    const std::vector<MetricEvaluation>& getMetricEvaluations() const;

private:
    std::string m_contributor;
    std::string m_fileId;
    int m_lineStart;
    int m_lineEnd;
    std::vector<MetricEvaluation> m_evaluations;
};
```

2.1.3 PaymentManager Class

The `PaymentManager` class handles payment tracking within a license:

```
class PaymentManager {
public:
    explicit PaymentManager(const std::string& walletAddress);

    bool recordPayment(const CodeContribution& contribution, double amount);
    double getTotalPaymentsForContributor(const std::string& contributor) const;
    std::string generatePaymentReport() const;

    // Getters
    const std::string& getWalletAddress() const;
    const std::unordered_map<std::string, double>& getPayments() const;

private:
    std::string m_walletAddress;
    std::unordered_map<std::string, double> m_payments;
};
```

Chapter 3

Metrics Evaluation

3.1 Metrics Overview

The CCSL system evaluates code quality using six distinct metrics:

3.2 Metrics Structure

3.2.1 MetricType Enumeration

The metrics are defined by the `MetricType` enumeration:

```
enum class MetricType {
    IMPACT,
    SIMPLICITY,
    CLEANNESS,
    COMMENT,
    CREDITABILITY,
    NOVELTY
};
```

3.2.2 MetricEvaluation Structure

Each evaluation of a metric is represented by the `MetricEvaluation` structure:

```
struct MetricEvaluation {
    MetricType type;
    double value;
    std::string rationale;
};
```

3.2.3 MetricEvaluator Interface

All metric evaluators implement the `MetricEvaluator` interface:

```
class MetricEvaluator {
public:
    virtual ~MetricEvaluator() = default;

    virtual MetricEvaluation evaluate(const std::string& code) const = 0;
    virtual std::string getDescription() const = 0;
    virtual MetricType getType() const = 0;
};
```

3.3 Implementation Details

Each metric evaluator analyzes code to provide a score between 0.0 and 1.0:

3.3.1 Impact Evaluator

Analyzes function calls and control structures to determine how much impact the code has on the overall functionality.

3.3.2 Simplicity Evaluator

Examines factors like line length, nesting depth, and symbol density to evaluate how easily the code can be understood.

3.3.3 Cleanness Evaluator

Checks formatting consistency, indentation, and bracket style to evaluate the code's cleanliness.

3.3.4 Comment Evaluator

Analyzes comment density, quality, and relevance to assess documentation value.

3.3.5 Creditability Evaluator

Looks for evidence of testing, documentation, and external references to establish credibility.

3.3.6 Novelty Evaluator

Identifies advanced language features, design patterns, and algorithm analysis to measure innovation.

Metric	Range	Description
Impact	0.0 - 1.0	Measures the gravity effect towards a particular line in the overall function of the program
Simplicity	0.0 - 1.0	Measures purity of syntactic, semantic, and pragmatic quality to be easily digested by programmers
Cleanness	0.0 - 1.0	Measures proper formatting and subsymbolic and symbolic notation
Comment	0.0 - 1.0	Measures quality of non-opinionated statements with no syntactic sugar
Creditability	0.0 - 1.0	Measures evidence that technique is compatible with architecture requirements
Novelty	0.0 - 1.0	Measures creative and exotic approach to problem-solving

Table 3.1: CCSL Metrics Summary

Chapter 4

Payment Integration

4.1 Bitcoin Integration

The payment integration component enables Bitcoin micropayments for code contributions through the following classes:

4.1.1 BitcoinPaymentManager Class

Manages Bitcoin transactions:

```
class BitcoinPaymentManager {
public:
    explicit BitcoinPaymentManager(const std::string& apiKey);

    bool initialize();
    std::future<std::string> sendPayment(
        const std::string& sourceWallet,
        const std::string& destinationWallet,
        double amount,
        const std::string& contributionId,
        PaymentVerificationCallback callback
    );
    bool verifyPayment(const std::string& transactionId);
    std::vector<PaymentTransaction> getTransactions() const;
    std::vector<PaymentTransaction> getTransactionsForContribution(
        const std::string& contributionId) const;

private:
    std::string m_apiKey;
    std::vector<PaymentTransaction> m_transactions;
};
```

4.1.2 PaymentSubscription Class

Manages recurring payment subscriptions:

```
class PaymentSubscription {
public:
    PaymentSubscription(
        const std::string& contributorId,
        const std::string& walletAddress,
        int subscriptionPeriod
    );

    bool processPayment(BitcoinPaymentManager& paymentManager, double amount);
    bool isPaymentDue() const;

    // Getters
    std::string getContributorId() const;
```

```
    std::string getWalletAddress() const;
    int getSubscriptionPeriod() const;
    std::chrono::system_clock::time_point getNextPaymentDate() const;

private:
    std::string m_contributorId;
    std::string m_walletAddress;
    int m_subscriptionPeriod;
    std::chrono::system_clock::time_point m_nextPaymentDate;
};
```

4.1.3 RecurringPaymentManager Class

Manages multiple payment subscriptions:

```
class RecurringPaymentManager {
public:
    explicit RecurringPaymentManager(BitcoinPaymentManager& paymentManager);

    void addSubscription(const PaymentSubscription& subscription);
    bool removeSubscription(const std::string& contributorId);
    int processDuePayments();
    const std::vector<PaymentSubscription>& getSubscriptions() const;

private:
    BitcoinPaymentManager& m_paymentManager;
    std::vector<PaymentSubscription> m_subscriptions;
};
```

4.2 Payment Flow

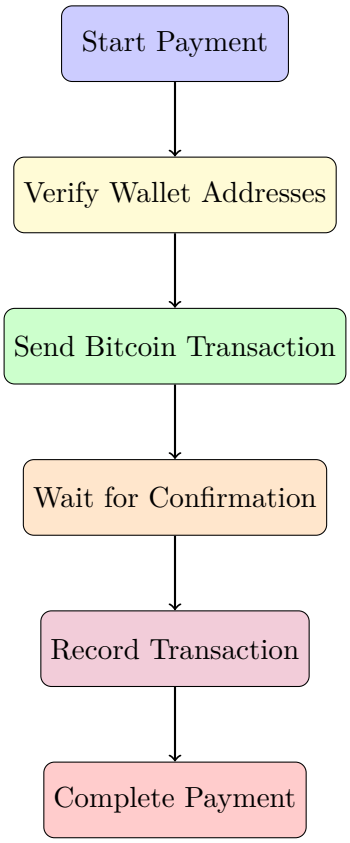

Figure 4.1: Payment Flow Diagram

Chapter 5

Usage Examples

5.1 Basic Usage Example

The following example demonstrates basic usage of the CCSL system:

```cpp
#include <ccsl/license.hpp>
#include <iostream>

using namespace ccsl;

int main() {
    // Create a license
    License license("Example Project", "CCSL-EXAMPLE-2025");

    // Register a contribution
    CodeContribution contribution("Alice Smith", "main.cpp", 1, 100);

    // Add metric evaluations
    MetricEvaluation impact;
    impact.type = MetricType::IMPACT;
    impact.value = 0.85;
    impact.rationale = "High impact code";

    contribution.addMetricEvaluation(impact);

    // Register the contribution
    license.registerContribution(contribution);

    // Record a payment
    license.getPaymentManager().recordPayment(contribution, 0.001);

    // Display license info
    std::cout << license.getLicenseInfo() << std::endl;

    return 0;
}
```

5.2 Metrics Evaluation Example

This example demonstrates how to evaluate code quality using the metrics system:

```cpp
#include <ccsl/metrics.hpp>
#include <iostream>

using namespace ccsl;

int main() {
    // Create code to evaluate
    std::string code = R"(
```

```
        // Calculate factorial
        int factorial(int n) {
            if (n <= 1) return 1;
            return n * factorial(n-1);
        }
    )";

    // Create metrics evaluator
    MetricsEvaluator evaluator;

    // Evaluate the code
    auto evaluations = evaluator.evaluateAll(code);

    // Print results
    for (const auto& eval : evaluations) {
        std::cout << "Metric: ";
        switch (eval.type) {
            case MetricType::IMPACT:       std::cout << "Impact"; break;
            case MetricType::SIMPLICITY:   std::cout << "Simplicity"; break;
            case MetricType::CLEANNESS:    std::cout << "Cleanness"; break;
            case MetricType::COMMENT:      std::cout << "Comment"; break;
            case MetricType::CREDITABILITY: std::cout << "Creditability"; break;
            case MetricType::NOVELTY:      std::cout << "Novelty"; break;
        }
        std::cout << ", Value: " << eval.value << std::endl;
        std::cout << "Rationale: " << eval.rationale << std::endl;
    }

    // Calculate overall value
    double value = evaluator.calculateValue(code);
    std::cout << "Overall value: " << value << std::endl;

    return 0;
}
```

5.3 Bitcoin Payment Example

This example demonstrates how to send Bitcoin payments for code contributions:

```
#include <ccsl/payment.hpp>
#include <iostream>

using namespace ccsl;

int main() {
    // Create payment manager
    BitcoinPaymentManager manager("api-key-123");

    // Initialize
    if (!manager.initialize()) {
        std::cerr << "Failed to initialize payment manager" << std::endl;
        return 1;
    }

    // Set up payment callback
    auto callback = [](const PaymentTransaction& tx, bool success) {
        if (success) {
            std::cout << "Payment successful: " << tx.amount << " BTC" << std::endl;
        } else {
            std::cerr << "Payment failed" << std::endl;
        }
    };

    // Send payment
    std::future<std::string> future = manager.sendPayment(
```

```
        "1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa",  // source
        "3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy",  // destination
        0.0005,  // amount
        "contribution-123",  // contribution ID
        callback
    );

    // Wait for result
    std::string txId = future.get();
    std::cout << "Transaction ID: " << txId << std::endl;

    return 0;
}
```

Chapter 6

Conclusion

6.1 Summary

The Chandra Credit Software License (CCSL) system provides a comprehensive framework for integrating code quality metrics with cryptocurrency micropayments. By objectively measuring code quality and enabling fair compensation, CCSL aims to create a more sustainable and equitable software development ecosystem.

6.2 Future Work

Future versions of the CCSL system will include:

- Integration with more cryptocurrencies beyond Bitcoin
- Advanced machine learning algorithms for metrics evaluation
- A web-based dashboard for managing licenses and payments
- Integration with popular version control systems
- Mobile applications for payment tracking and verification

6.3 Final Thoughts

The CCSL system represents a significant step toward recognizing and rewarding the true value of code contributions. By combining objective quality metrics with cryptocurrency micropayments, it offers a new model for software licensing that benefits both contributors and users.

Appendix A

License Text

```
# Chandra Credit Software License (CCSL)
Version 0.1, May 2025

## 1. Introduction

This Chandra Credit Software License ("License") is a legal agreement between any person or entity ("Licensee") and the author(s) of the software ("Licensor") regarding the use of software that is distributed under this License. By using, copying, modifying, or distributing the software, you accept and agree to be bound by the terms of this License.

## 2. Definitions

a. "Software" refers to the source code, object code, documentation, and any other materials provided by the Licensor under this License.

b. "Contribution" refers to any modification, enhancement, bug fix, or other work submitted to the Licensor for inclusion in the Software.

c. "Contributor" refers to any individual or entity that creates a Contribution.

d. "Credit Score" refers to the numerical value assigned to a Contribution based on the evaluation of its quality metrics.

e. "Bitcoin Address" refers to a unique identifier used for sending and receiving Bitcoin payments.

## 3. Grant of License

Subject to the terms and conditions of this License, the Licensor hereby grants to the Licensee a worldwide, non-exclusive license to:

a. Use, reproduce, modify, and distribute the Software;

b. Prepare derivative works based on the Software;

c. Publicly display and perform the Software and derivative works.

## 4. Credit Metrics System

The Software shall include a Credit Metrics System that evaluates the quality of Contributions using the following metrics:

a. Impact: Measures the gravity effect towards a particular line in the overall function of the program (0.0-1.0).

b. Simplicity: Measures purity of syntactic, semantic, and pragmatic quality to be easily digested by programmers (0.0-1.0).

```

```
c. Cleanness: Measures proper formatting and subsymbolic and symbolic notation (0.0-1.0).

d. Comment: Measures quality of non-opinionated statements with no syntactic sugar (0.0-1.0).

e. Creditability: Measures evidence that technique is compatible with architecture requirements (0.0-1.0).

f. Novelty: Measures creative and exotic approach to problem-solving (0.0-1.0).

## 5. Credit Attribution

a. The Software shall maintain a record of all Contributors and their respective Contributions.

b. Each Contribution shall be evaluated using the Credit Metrics System, resulting in a Credit Score.

c. Credit Scores shall be publicly accessible and associated with the Bitcoin Address of the respective Contributor.

## 6. Payment Obligations

a. Any entity that uses the Software in a commercial product or service shall make Bitcoin payments to Contributors based on their Credit Scores according to the following formula:

   Payment = Credit Score    Lines of Code    Base Rate

   where Base Rate is determined by the Licensor and published with the Software.

b. Payments shall be made to the Bitcoin Address associated with each Contributor.

c. Payment frequency shall be at least quarterly for ongoing commercial use.

## 7. Redistribution Requirements

a. Redistributions of the Software in source code form must retain this License text, the list of Contributors, their Credit Scores, and their Bitcoin Addresses.

b. Redistributions in binary form must reproduce this License text, the list of Contributors, their Credit Scores, and their Bitcoin Addresses in the documentation and/or other materials provided with the distribution.

c. Modified versions of the Software must carry prominent notices stating that the Software has been modified, and provide a means to access the modified source code.

## 8. No Warranty

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 9. Termination

This License automatically terminates if the Licensee fails to comply with any of its terms and conditions. Upon termination, the Licensee must cease all use, reproduction, modification, and distribution of the Software.

## 10. Miscellaneous

```

```
a. This License constitutes the entire agreement between the parties with respect to
    the use of the Software.

b. This License shall be governed by and construed in accordance with the laws of the
     jurisdiction in which the Licensor resides.

c. If any provision of this License is held to be unenforceable, such provision shall
     be reformed only to the extent necessary to make it enforceable.
```

Listing A.1: CCSL License Text