Programming Hints for Smart Contracts on the Fly

Jemin Andrew Choi
Department of Computer Science
University of Toronto
choi@cs.toronto.edu

ABSTRACT

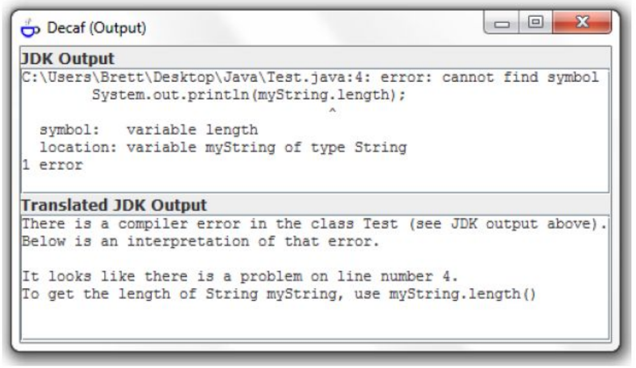
Despite growing security risks on the blockchain, there is little literature on helping educate smart contract developers about topics on blockchain security. We present VSolythesis, a system designed to give programming hints on the fly for developers to navigate and understand different security checks available for smart contracts. We also suggest some methods for evaluating this system for potential users and discuss some limitations of applying traditional pedagogical techniques in a specialized domain such as blockchain.

1 Introduction

There have been great strides made in the field of traditional Computer Science education, but there have been fewer efforts to apply these principles in specialized domains, such as blockchain [3]. For instance, there are over 21,000 queries on the smart contract language, Solidity, on StackOverflow, which suggests the need for a better learning framework for developers and students alike [1]. Although there have been efforts to unify blockchain and Human-Computer Interaction concepts and integrate the blockchain ecosystem from an educational perspective, there has not been much research done in tying the actual art of blockchain programming with Computer Science pedagogy [3, 4].

There is also a growing demand for robust systems that interact with existing blockchains. In Ethereum, a popular blockchain ledger, developers interact with smart contracts, which are programs that encode rules for transactions on the blockchain [8]. Although smart contracts can empower developers to create fully automated transactions and specify sophisticated rules, it can introduce equally complicated bugs that can have extremely large fiscal consequences. This is particularly exemplified by the abundance of securities flaws that were exploited by hackers in Ethereum [8]. For instance, there have been incidents like the DAO hack and the BEC hack that were introduced into Ethereum due to a lack of security features in smart contracts [9, 10]. In addition to millions of dollars in financial losses, these security flaws undermine the trust within the blockchain system, which magnifies the intangible costs of these hacks.

Although there has been active work in making smart contracts more robust through static analysis [11] and run-time analysis [15], another approach is to create a better system to educate developers on these security flaws. There have been studies done on consumer useability of blockchain, but there is less literature on how to approach blockchain useability from a developer's perspective [5, 6, 7]. Similarly, the relative novelty of blockchain as an academic field has resulted in less emphasis in developing formal methods in pedagogy. Hence, the motivation for this project stems from the need to unify the blockchain developers' experiences with Computer Science education.



```
C:\Users\Brett\Desktop\Java\Test.java:4: error: cannot find symbol
        System.out.println(myString.length);
                                   ^
  symbol:   variable length
  location: variable myString of type String
1 error
```

```
There is a compiler error in the class Test (see JDK output above).
Below is an interpretation of that error.

It looks like there is a problem on line number 4.
To get the length of String myString, use myString.length()
```

Figure 1: Decaf, a compile-time helper that provides helpful and interpretable compiler messages [2].

There is also notable research being done in compile-time analysis to help programming novices learn proper coding practices and reduce errors. For instance, Decaf, shown in Figure 1, is a system that provides helpful compiler error messages that can encourage and guide students towards correcting syntax errors [2]. Similarly, there have been studies to show how hints can be shown to students and users in a constructive manner [13, 14]. Naturally, it is interesting to question whether these principles can also be applied into the context of Blockchain, and whether it can be used to help educate students and developers about potential security flaws in smart contracts.

In this paper, we will outline a system, called VSolythesis, that combines elements of interactive hints and code suggestions with smart contracts, and suggest how this system can be evaluated for effectiveness. VSolythesis is a VS Code extension written in Typescript, designed to give real-time hints and feedback to users. It leverages a runtime analysis tool called Solythesis [15], to instrument potentially faulty smart contracts, and suggest hints based on the instrumented code. The ultimate objective of this study is to observe whether interactive programming tools help or hinder the learning experience in the blockchain domain.

2 Related Work

In this section, we examine two systems from different domains, and we will unify concepts from these systems to create a new system to suggest programming hints for smart contract developers.

2.1 iSnap

iSnap is a programming environment based on Scratch that provides learning features such as logging and on-demand hints for students [14]. iSnap is able to log student submissions to a database, where it can be used to generate automatic and adaptive hints for other students.

In studies that evaluate the effectiveness of the iSnap system, students found textual hints and explanations to be particularly useful in understanding bugs in their code [13, 14]. On the other hand, features such as the self-explanation prompts had mixed reviews, with some students noting that this feature confused them even more. Some students also noted that hints indicating progress and positive reinforcement may have been nice, as the system only responded when there was an error in the code, which suggests a higher level of interactivity is needed between the system and the user.

2.2 Solythesis

Solythesis is a runtime validation tool that instruments smart contracts, written in Solidity, based on user-specified invariants [15]. The instrumented smart contract will reject transactions that violate various safety checks, such as integer underflow re-entrancy flaws that have previously been the source of various hacks [9, 10].

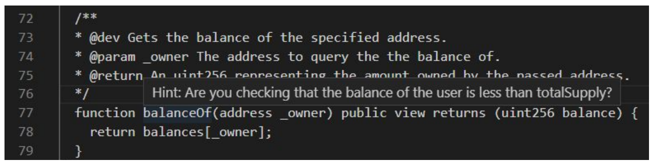
Although there are other static validation tools for smart contracts [11, 12], we chose to use Solythesis as it is able to insert run-time checks, which are often more powerful and flag fewer false positives compared to static analysis. In addition, the computational overhead of inserting run-time checks is not considered to be a significant limitation in this domain, as users do not benefit from increases in execution efficiency in an educational setting.

3 Design

In this section, we outline some features of VSolythesis and discuss some design decisions that were made to integrate hints with smart contract programming.

3.1 VSolythesis

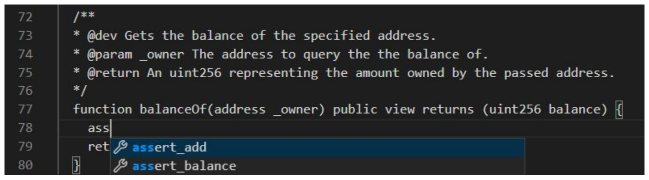
VSolythesis combines elements from iSnap and other hint-based tools to provide users a nudge towards error correction [2, 14]. It takes advantage of the runtime analysis of Solythesis, and generates instrumented code that fixes security errors from the source code based on user-specified invariants. Then, VSolythesis shows the user inline hints based on the differences between the original source code and the instrumented source code. Figure 2 shows an example of a generated inline hint on the fly for the BEC contract.



```
/**
* @dev Gets the balance of the specified address.
* @param _owner The address to query the the balance of.
* @return An uint256 representing the amount owned by the passed address.
*/         Hint: Are you checking that the balance of the user is less than totalSupply?
function balanceOf(address _owner) public view returns (uint256 balance) {
  return balances[_owner];
}
```

Figure 2: Programming hint for a function in the BEC contract in Solidity [17].

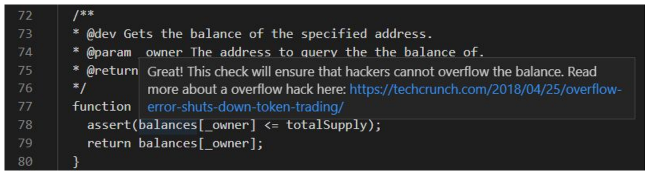
After VSolythesis generates a hint for the user, it prompts the user to insert an assertion or some check to patch a specific security flaw. The user can correct the error by typing the assert command, which prompts the user with a choice of assertions, as seen on Figure 3. When the user selects a certain type of assertion, the system auto-completes the rest of the code. This reduces some of the boilerplate code that is required for some checks, and it helps users focus on the semantics of the code, rather than the specific syntax of these checks.



```
/**
* @dev Gets the balance of the specified address.
* @param _owner The address to query the the balance of.
* @return An uint256 representing the amount owned by the passed address.
*/
function balanceOf(address _owner) public view returns (uint256 balance) {
  ass
  ret assert_add
}     assert_balance
```

Figure 3: A list of assertion prompts from VSolythesis.

VSolythesis relies on a lexical analyzer (IntelliSense) and a state machine to determine whether the user requires more hints or is in the final, correct state [16]. We developed the system with the student feedback from [13] in mind, and we wanted to provide some sort of feedback indicating the user's progress on the goal at hand. Since we do not have access to as much data as traditional programming languages or applications, we generate hints based on the standards specified by the smart contract and on the user-specified invariants. If the user enters an assertion that does not fit the current context, then the system prompts the user to try a different assertion. If the user enters the correct assertion, the system reinforces the answer with an encouraging message and supplies a real-life example of the security exploit, if applicable, and an image of this is shown on Figure 4.



```
/**
* @dev Gets the balance of the specified address.
* @param  owner The address to query the the balance of.
* @return
*/
function
  assert(balances[_owner] <= totalSupply);
  return balances[_owner];
}
```

Great! This check will ensure that hackers cannot overflow the balance. Read more about a overflow hack here: https://techcrunch.com/2018/04/25/overflow-error-shuts-down-token-trading/

Figure 4: Inline message denoting that the user has inserted the correct assertion. The system also provides a link to a real-world example of this security exploit.

4 Proposed Method and Evaluation

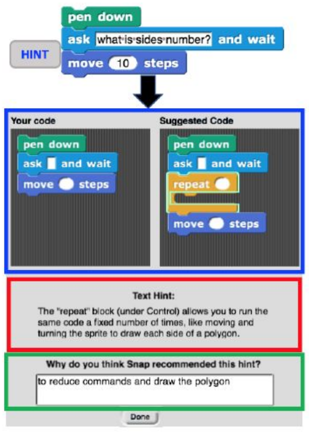
In this section, we propose a method to evaluate the effectiveness of VSolythesis on smart contract programmers. The study would focus on students or developers who have basic familiarity with smart contract programming and Solidity.

The proposed experiment would have two groups of users try to identify and patch security flaws in different smart contracts in one hour. The smart contracts would be pulled from different contract standards, such as ERC20 (BEC) and ERC1202 (Vote) [17]. Both groups would have access to the source code, with security flaws, a set of invariants, and invalid transaction test cases that suggest what these flaws could be. The control group would also have access to the instrumented code, but without the VSolythesis extension to correct the source code. The test group would only have access to the extension to help guide their decisions in patching up the source code. Our hypothesis is that the VSolythesis extension will help users identify and correct security flaws more quickly than the control group. We also think that the system will help improve the learning experience by providing interactive and real-time feedback.

The experiment would also have a qualitative evaluation as well, such as the one conducted in [13]. After the experiment, the users would fill out some questions to test their understanding of various security flaws in the smart contracts. The questions can be used as an indicator to observe whether the system aided or hindered the users' understanding of these invariants. In addition, it can be used to aggregate feedback about how interactive feedback systems can be better integrated in a code editor, specifically for smart contract programming.

5 Discussion

In this section, we discuss and analyze some facets of applying traditional pedagogical techniques in blockchain, some limitations of the system, and potential directions for future work.



Text Hint:

The "repeat" block (under Control) allows you to run the same code a fixed number of times, like moving and turning the sprite to draw each side of a polygon.

Why do you think Snap recommended this hint?

to reduce commands and draw the polygon

Figure 5: The iSnap interface, where the user is shown the suggested solution in the blue box and a hint in the red box [13].

There is ample literature that suggests the effectiveness of adaptive hints, and these studies served as motivation for this paper. In previous studies such as [13, 14], the authors apply pedagogical techniques such as automated hints with self-explanation prompts to help students' performance in introductory Computer Science courses. Figure 5 shows an example of this interface. In [22], the authors leveraged a social recommender system that directly stores students' submissions to suggest explanations and relevant code snippets. In [21], the authors implemented a machine learning algorithm to crowd-source student explanations and found it to be effective for primary-level math questions. Although these works were inspiration for this work, we outline some differences between introductory Computer

Science and blockchain that makes a direct implementation or comparison difficult.

5.1 Decentralization and Anonymity

One of the core tenets of blockchain is decentralization and anonymity, which may impact the users' willingness to share any data over a database [24]. The works listed above make heavy use of student submissions and have access to vast amounts of user data. For our study, it appears that we do not have the same luxury, nor the potential to gather much information about our users. The relative anonymity of some blockchain developers or hacker groups may be contributing to the difficulty in establishing principles in blockchain education.

In addition, perhaps a traditional approach is not suitable for educating students on cybersecurity and blockchain topics, because we do not know whether a system is secure until someone is able to hack into it. There has been literature showing that a more hands-on approach, such Capture-The-Flag or hacking competitions may be more effective in teaching abstract and transferable skills, such as reverse engineering [25]. And consequently, these transferable skills may benefit students more in the long run. Ostensibly, this is different in scope from this paper, as we chose to educate developers more about the mechanics and semantics of programming smart contracts. However, exploring different hacker cultures and cybersecurity competitions seems to be fruitful for future work, as these domains appear to be much closer to blockchain than introductory Computer Science topics.

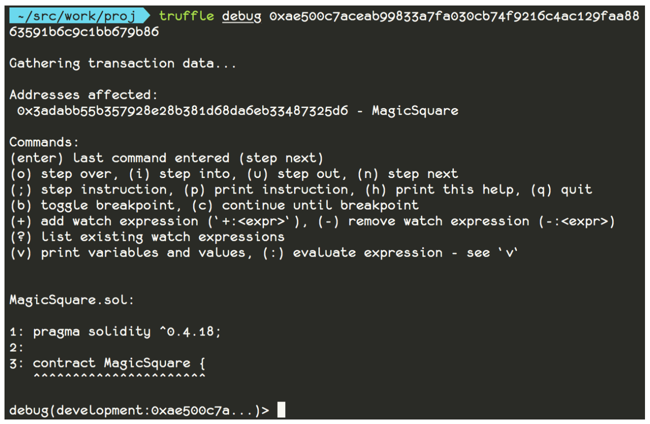
5.1 Limitations

A limitation of VSolythesis is that the suggested code or instrumented code may not be idiomatic or understandable for novice programmers. The system attempts to mitigate this through the auto-complete feature, which helps reduce the amount of boilerplate code that needs to be written by the user. However, it may still be hard for users to grasp why the code works, line by line. Some potential ideas to further mitigate this is discussed in the Future Work section.

Another potential limitation of this work is that applications in blockchain may not be easily generalizable in other domains, because smart contracts are rigidly defined by standards. This allows hints to be easily generated using lexical analysis for smart contracts, but this is not necessarily true for other applications. On the other hand, we hope that this can be a starting point for further research on topics relevant to blockchain education.

5.2 Future Work

Future work for this system is to integrate a debugger into the extension so students can actually execute some transactions on the smart contract and observe effects line by line. The existing literature on how novices approach debuggers, and some debugging techniques can be applied to help design a debugging tool for novice smart contract programmers[18, 19]. In addition, there are some newer debuggers for Solidity, such as Truffle as seen on Figure 6, that could be used as a baseline for debugging tools [20]. However, there is still lots of potential to extend these tools to foster safer smart contract programming practices.



```
 ~/src/work/proj  truffle debug 0xae500c7aceab99833a7fa030cb74f9216c4ac129faa88
63591b6c9c1bb679b86

Gathering transaction data...

Addresses affected:
 0x3adabb55b357928e28b381d68da6eb33487325d6 - MagicSquare

Commands:
(enter) last command entered (step next)
(o) step over, (i) step into, (u) step out, (n) step next
(;) step instruction, (p) print instruction, (h) print this help, (q) quit
(b) toggle breakpoint, (c) continue until breakpoint
(+) add watch expression (`+:<expr>`), (-) remove watch expression (-:<expr>)
(?) list existing watch expressions
(v) print variables and values, (:) evaluate expression - see `v`


MagicSquare.sol:

1: pragma solidity ^0.4.18;
2:
3: contract MagicSquare {
   ^^^^^^^^^^^^^^^^^^^^^^


debug(development:0xae500c7a...)> █
```

Figure 6: Truffle Debugger interface. There is still ample work to be done in enhancing the useability of smart contract development tools.

Another direction for future work includes suggesting idiomatic code for users. Since the instrumented code may not always be understandable for novices, it would be helpful to automatically generate novice-friendly code. There has been some research done in learner-sourcing explanations using machine learning algorithms, which can be applied to generating code snippets as well [21]. Similarly, there is potential to build up a database for user-submitted code to generate suggestions as well, such as in the iSnap database [14]. However, this seems to be a challenge in the status quo as the blockchain community is still small and comparatively fewer developers work with Solidity, and we hope that more developers will work with researchers on this front in the future.

6 Conclusion

We presented a system, VSolythesis, that suggests programming hints for smart contract development based on specified invariants. We combined approaches from automatic and adaptive hint generation and run-time analysis that encourages developers to write safer smart contracts. Despite some limitations in terms of data collection, we analyze and discuss how the future of blockchain education might look like, and we hope that this will help spur future work in the intersection of Computer Science education and blockchain security.

REFERENCES

[1] [n.d.]. StackExchange. https://stackexchange.com/search?q=%5Bsolidity%5D

[2] Brett A Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. Computer Science Education 26, 2-3 (2016), 148–175.

[3] Guang Chen, B. Xu, M. Lu, and N. Chen. 2018. Exploring blockchain technology and its potential applications for education. Smart Learning Environments. 5.

[4] Chris Elsden, Bettina Nissen, Karim Jabbar, Reem Talhouk, Caitlin Lustig, Paul Dunphy, Chris Speed, and John Vines. 2018. HCI for Blockchain: Studying, Designing, Critiquing and Envisioning Distributed Ledger Technologies. In Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (CHI EA '18). Association for Computing Machinery, New York, NY, USA, Paper W28, 1–8.

[5] Karim Jabbar and Pernille Bjørn. 2019. Blockchain Assemblages: Whiteboxing Technology and Transforming Infrastructural Imaginaries. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19). Association for Computing Machinery, New York, NY, USA, Paper 266, 1–13.

[6] Chris Elsden, Tom Feltwell, Shaun Lawson, and John Vines. 2019. Recipes for Programmable Money. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19). Association for Computing Machinery, New York, NY, USA, Paper 251, 1–13.

[7] Chris Elsden, Arthi Manohar, Jo Briggs, Mike Harding, Chris Speed, and John Vines. 2018. Making Sense of Blockchain Applications: A Typology for HCI. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). Association for Computing Machinery, New York, NY, USA, Paper 458, 1–14.

[8] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151 (2014), 1–32.

[9] John Biggs. 2018. Overflow error shuts down token trading. https://techcrunch.com/2018/04/25/overflow-error-shuts-down-token-trading/

[10] Phil Daian. 2016. Analysis of the DAO exploit. http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/.

[11] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bunzli, and Martin Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). ACM,

[12] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. Zeus: Analyzing safety of smart contracts. NDSS.

[13] Samiha Marwan, Joseph Jay Williams, and Thomas Price. 2019. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning. In Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19). Association for Computing Machinery, New York, NY, USA, 61–70.

[14] Thomas W. Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In Proceedings of the ACM Technical Symposium on Computer Science Education.

[15] A. Li, J. Choi, and F. Long. 2019. Securing Smart Contract On The Fly. arXiv preprint arXiv:1911.12555.

[16] Microsoft. 2016. IntelliSense in Visual Studio Code. https://code.visualstudio.com/docs/editor/intellisense

[17] [n. d.]. Ethereum (ETH) Blockchain Explorer. https://etherscan.io/.

[18] L. Gugerty and G. Olson. 1986. Debugging by skilled and novice programmers. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86). Association for Computing Machinery, New York, NY, USA, 171–174.

[19] Amy J. Ko and Brad A. Myers. 2004. Designing the whyline: a debugging interface for asking questions about program behavior. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04). Association for Computing Machinery, New York, NY, USA, 151–158.

[20] Mayowa Tudonu. 2018. Debugging Smart Contracts with Truffle Debugger: A Practical Approach. (November 2018). https://hackernoon.com/debugging-smart-contracts-with-truffle-debugger-a-practical-approach-f56bf0600736

[21] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z. Gajos, Walter S. Lasecki, and Neil Heffernan. 2016. AXIS: Generating Explanations at Scale with Learnersourcing and Machine Learning. In Proceedings of the Third (2016) ACM Conference on Learning @ Scale (L@S '16). Association for Computing Machinery, New York, NY, USA, 379–388.

[22] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What would other programmers do: suggesting solutions to error messages. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10). Association for Computing Machinery, New York, NY, USA, 1019–1028.

[23] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. 2016. Learnersourcing Personalized Hints. In Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW '16). Association for Computing Machinery, New York, NY, USA, 1626–1636.

[24] Jiajun Xin, Pei Huang, Lei Chen, Xin Lai, Xiuyu Zhang, Wulu Li, and Yongcan Wang. 2019. WaterCarver: Anonymous Confidential Blockchain System based on Account ModelIACR Cryptology ePrint Archive, 2019, 1265.

[25] Vitaly Ford, Ambareen Siraj, Ada Haynes, and Eric Brown. 2017. Capture the Flag Unplugged: an Offline Cyber Competition. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). Association for Computing Machinery, New York, NY, USA, 225–230.