ファジングツール AFL の利用を支援するツール Fuzz4B によるファジング教育の試み

宮木 龍　吉田 則裕　藤原 賢二　高田 広章

近年，ソフトウェアの開発規模が飛躍的に増大し，ソフトウェアテストの重要性が一層高まっている．それに伴い，ソフトウェアテストの自動化に対する需要が高まっている．自動化が可能なソフトウェアテストの 1 つとして，ファジングが注目されている．ある大学の講義においてファジングを講義の主な題材として取り上げ，Fuzz4B を通じてファジングを利用する演習を実施した結果について述べる．

In recent years, software development has increased dramatically, and the importance of software testing has become even more significant. As a result, the demand for software testing automation has been increasing. Fuzzing has been attracting attention as one of the software tests that can be automated. This paper describes the results of an exercise to use fuzzing through Fuzz4B in a university course.

1 はじめに

近年，ソフトウェアの開発規模が飛躍的に増大し，ソフトウェアテストの重要性が一層高まっている [7]．それに伴い，ソフトウェアテストの自動化に対する需要が高まっている．自動化が可能なソフトウェアテストの 1 つとして，ファジングが注目されている．ファジングは，ファズと呼ばれるデータを生成してテスト対象のソフトウェアに入力し，挙動を監視することで不具合を自動で検出するテスト手法である [2]．ファジングを自動で行うツールをファジングツールと呼ぶ．ファジングによって不具合の低減やテストにかかる労力の削減といった効果が得られる [2]．しかし，実際にファジングを導入して活用しているのは大手企業に限られ，活用事例は少ないというのが現状である [2]．

また，一般的なソフトウェアテストに関する教育については数多くの既存研究 [3] [12] が存在する．例えば，高澤らは，ソフトウェアテストのゲーミフィケーションを目的として，テスト貢献度をテスト担当者間で共有する教育用テストツールを提案している [9]．また，森らはコンソーシアム型共同研究を通じたソフトウェアテスト教育について報告している [6]．ファジングの教育を行うためには，ファジングや関連する技術について演習を通して学習する必要があると考えられるが，著者らが知る限りファジングの教育を試みた既存研究は存在しない．

我々はこれまでに，ファジングツールの導入の敷居を下げ，ファジングの活用事例を多くすることを目指してファジングツール利用支援ツール **Fuzz4B** を開発してきた [4] [5]．**Fuzz4B** はオープンソースソフトウェアであり，GitHub リポジトリ †1 から利用可能である．我々は，ファジングを大学の講義における題材として取り上げ，学生がファジングについて学ぶ機会を設ければ，ファジングの活用事例の増加に繋がると考えた．そこで我々は，ある大学の講義においてファジングを講義の主な題材として取り上げ，**Fuzz4B** を通じてファジングを利用する演習を実施した．本論文では，講義内容と理解度確認テストやアンケートの結果について述べ，受講生のファジングに対する理解

Fuzzing Education Using Fuzz4B: A Support Tool for Fuzzing with AFL
Ryu Miyaki, Norihiro Yoshida, Hiroaki Takada, 名古屋大学, Nagoya University.
Kenji Fujiwara, 東京都市大学, Tokyo City University.

†1 https://github.com/Ryu-Miyaki/Fuzz4B

度や関心を持てたかどうかについて考察する．

2 ファジング支援ツール Fuzz4B

本章では，**Fuzz4B** の持つ機能のうち，演習で使用した **AFL GUI**，**DD**，**ReproCrash** という 3 つの機能について説明する．

2.1 AFL GUI

これまでに開発された数多くのファジングツールのうち，代表的なものとして **AFL**[†2] が挙げられる．**AFL** は，オープンソースのファジングツールで，C 言語で記述されたプログラムを対象としている．**AFL** は，これまでに数多くの脆弱性を発見した運用実績があり，数多くのファジングツールが **AFL** を拡張することで開発されている [1] [8] [10]．以上の理由から，**Fuzz4B** は **AFL** を対象とし，その利用を支援する．

AFL を起動してファジングを行う際は，以下に示す 3 つの手順が必要である [4]．そのため，**AFL** の利用経験がないユーザは **AFL** を起動するために **AFL** について学習する必要が生じ，コストがかかるという問題がある [4]．

手順 1 専用のコンパイラを使用してテスト対象ソフトウェアの実行ファイルを作成する．

手順 2 **AFL** が最初にテスト対象ソフトウェアに入力するデータとして使用する初期入力を用意する．

手順 3 **AFL** を起動するためのコマンドを適切に実行する．

AFL GUI は，上記の問題を軽減するために，**AFL** の GUI を提供する機能である．**Fuzz4B** は，**AFL GUI** によって，GUI を介して**手順 1**，**手順 2** が必要であることをユーザに伝える．また，**手順 3** は，**Fuzz4B** がユーザに代わって実行する．以上により，**Fuzz4B** はユーザが **AFL** について学習するコストを軽減する．

2.2 DD

AFL は，ファジングによってテスト対象ソフトウェアの不具合を検出した場合，不具合を起こすファズを出力する．しかし，**AFL** が出力したファズのサイズが大きい場合，そのファズのうち，不具合を起こす直接の原因となる箇所が分からないという問題がある．

DD は，上記の問題を軽減するために，デルタデバッギング [11] によって不具合を起こすファズのサイズを削減する機能である．デルタデバッギングは，ソフトウェアの不具合を起こすデータのサイズを，入力すると不具合を起こすという条件を保ったまま最小化するアルゴリズムである [11]．ここでの最小化とは，デルタデバッギングによって得られたデータの全てが不具合に関連していることであるとする [11]．**Fuzz4B** の **DD** は，デルタデバッギングによって不具合を起こすファズを最小化する作業を自動化する機能である．以上により，**Fuzz4B** は不具合を起こすファズから不具合の原因となる箇所を特定する作業を支援する．

2.3 ReproCrash

AFL は不具合を起こすファズ等を，1 つのディレクトリに格納して出力する．しかし，ユーザが **AFL** をデバッグで利用する際の流れは **AFL** のドキュメントで言及されておらず，**AFL** が出力するディレクトリ内部の詳細な情報も，**AFL** のドキュメントにおける出力についての説明で言及があるだけである．そのため，**AFL** の利用経験がないユーザは **AFL** が検出した不具合を再現する方法が分からないという問題がある [4]．

ReproCrash は，上記の問題を軽減するために，**AFL** が出力した任意のファズをテスト対象ソフトウェアに入力した状態で GDB[†3] を起動する機能である．GDB は，UNIX システムをはじめとした多くのシステムで動作するデバッガである．GDB を使用すれば，プログラムの実行を任意の箇所で停止することや，実行中の変数の値を確認することができる．**Fuzz4B** は，**ReproCrash** によって，**AFL** が出力

†2 http://lcamtuf.coredump.cx/afl/

†3 https://www.gnu.org/software/gdb/

した不具合を起こすファズをテスト対象ソフトウェアに入力した状態で GDB を起動する．ユーザは，起動した GDB を操作することで，不具合を再現することができる．以上により，**Fuzz4B** はユーザが不具合を再現する作業を支援する．

3 講義の内容

ある国立大学の情報工学系学科における 2021 年度の講義として，1 日あたり 180 分の講義を 3 日間実施した．感染症対策のため，Microsoft Teams を用いた遠隔講義を実施した．教員 2 名（うち一人は第二著者）およびティーチングアシスタント 1 名（第一著者）が担当し，学部 3 年生 3 名および 4 年生 1 名の計 4 名が受講した．

本講義の流れは，以下のとおりである．

1. 予習（ソフトウェアの信頼性，コーディング規約，信頼度成長曲線，ソフトウェアテスト，形式検証，コード生成ツール，N バージョンプログラミングなど）
2. ファジングの基礎に関する講義（assert 関数，ミューテーションに基づくファジング，**AFL**，カバレッジの計測，GDB など）
3. デルタデバッギングに関する講義
4. 理解度確認テスト 1（デルタデバッギング，ミューテーションに基づくファジング，GDB）
5. 宿題（逆ポーランド記法）
6. 演習 A: **Fuzz4B** のインストール・起動
7. 演習 B: デルタデバッギング with **Fuzz4B** チュートリアル
8. 演習 C: デルタデバッギング with **Fuzz4B** 演習
9. 演習 D: GDB チュートリアル
10. 演習 E: **Fuzz4B** + GDB チュートリアル
11. 理解度確認テスト 2（ファジング，デルタデバッギング，GDB）
12. レポート課題（**Fuzz4B** + GDB 演習）

講義日以前に，1 の内容を概説した A4 で合計 11 ページの資料を配布し，予習を行うよう指示した．予習の目的は，演習で使用される各技術を理解するために必要な用語の意味を理解することである．

2〜4 までは 1 日目に実施し，2 日目の予習として 5 の宿題を課した．6〜8 については，2 日目に実施し，3 日目は 9〜11 を実施した．レポート課題として 12 を課した．以降，各演習およびレポート課題について，それぞれ小節で述べる．

3.1 演習 A: Fuzz4B のインストール・起動

講義は遠隔で実施され，演習は受講者が各自で所持する PC を用いて実施することとした．受講者毎に使用する環境が異なると，トラブルが発生する原因になると考えた．そこで，**Fuzz4B** および **Fuzz4B** の利用に必須のソフトウェアをインストールした仮想マシンを事前に用意し，仮想マシンをエミュレータで実行することで，受講者が演習で使用する環境の統一を図った．講義では，エミュレータとして QEMU[†4] を使用した．

演習 A では，まず仮想マシンを起動できることを確認した．仮想マシンを起動させる手順をまとめた資料を受講者に配布し，資料の内容を受講者と共に確認しながら仮想マシンを起動できることを確認した．資料の内容通りの手順を実施しても仮想マシンを起動できなかった受講者については，個別に対応を行った．

次に，仮想マシン上で **Fuzz4B** の動作確認を行った．**Fuzz4B** の動作確認は，以下の手順で行った．

1. 事前に用意した，不具合を含むソフトウェアをコンパイルする．
2. 初期入力を作成する．
3. **AFL GUI** により，不具合を含むソフトウェアにファジングを実行し，不具合を検出する．
4. **DD** により，**AFL** が出力した不具合を起こすファズのサイズを削減する．
5. **ReproCrash** により，**AFL** が検出した不具合を再現する．

以上の手順をまとめた資料を受講者に配布し，資料の内容を受講者と共に確認しながら，**Fuzz4B** の動作確認を行った．

†4 https://www.qemu.org/

3.2 演習 B: デルタデバッギング with Fuzz4B チュートリアル

演習 B の目的は，**Fuzz4B** の **DD** を用いてデルタデバッギングを行い，欠陥修正を行う経験をしてもらうことである．演習 A と同様に，ハンズオンチュートリアル形式で各受講生の進捗を確認しながら実施し，1 名でも作業が遅れた受講生がいたら，その受講生が追いつくまで個別に指導を行った．

対象プログラムは，1 日目終了時に課した宿題において学習した逆ポーランド記法で式を計算するプログラムであり，初期入力 “1 2 + 3 * ”を与えファジングを行うとゼロ除算エラーが発生しクラッシュするように欠陥が埋め込まれている．デルタデバッギングを行うと，ゼロ除算を表す短いファズが出力され，受講生はゼロ除算が原因でクラッシュが発生していることがわかるようになっている．

演習の最後では，オペランドがゼロの場合の例外処理を対象プログラムに追記し，ファジングやデルタデバッギングにより得られた文字列でクラッシュが発生しないことを確認した．

3.3 演習 C: デルタデバッギング with Fuzz4B 演習

演習 C の目的は，演習 B と同様に **Fuzz4B** の **DD** を用いてデルタデバッギングを行い，欠陥修正を行うことである．演習 B ではハンズオンチュートリアル形式で各受講生の進捗を確認しながら実施したが，演習 C では当初は答えを教えず，受講生が自力で演習に取り組むように促した．ある程度，時間が経過したあとは，各受講生の進捗を確認しながらヒントを与えることで，行き詰まっている受講生を支援した．

対象プログラムは，入力された文字列を辞書順にソートし，出力するプログラムであり，入力された文字列において ‘(’ の後に ‘)’ が含まれていると，assert 文が実行されプログラムが停止するようになっている．

受講生には，assert 文が実行される条件を考えるように指示し，その後 assert 文を使用せずにエラーメッセージを出力したあとで exit 関数で終了するプログラムを変更するよう指示した．最後に，変更後プログラムを実行し，ファジングやデルタデバッギングにより得られた文字列でクラッシュが発生しないことを確認した．

3.4 演習 D: GDB チュートリアル

演習 D の目的は，演習 E やレポート課題で使用する GDB を使ったデバッグを経験してもらうことである．まずは，GDB を使った一般的なデバッグを経験してもらうことが目的であるため，**Fuzz4B** や **AFL** は使用しなかった．また，演習 B と同様にハンズオンチュートリアル形式で各受講生の進捗を確認しながら実施した．GDB にはステップ実行やウォッチなどさまざまな機能があるが，演習 E やレポート課題で主に必要な機能は停止行の確認とブレークポイントであるため，演習 D ではこれら 2 つの機能を主に扱った．ステップインやステップオーバ，ステップアウトについては，それらの説明を記載した補足資料を別途配布した．

対象プログラムは，既知のプログラムであった方が習熟しやすいと考え，演習 B と同じプログラムを対象プログラムとし，停止行の確認やブレークポイントの設定を行なった．

3.5 演習 E: Fuzz4B+GDB チュートリアル

演習 E の目的は，**Fuzz4B** の **ReproCrash** を用いることでデバッグを行い，欠陥修正を行うことである．演習 A や演習 B，演習 D と同様にハンズオンチュートリアル形式で各受講生の進捗を確認しながら実施した．

対象プログラムは，簡単な自動応答システムで，1〜3 の数値を入力すると，数値に応じたメッセージが出力されるプログラムであり，2 を入力するとセグメンテーションフォールトが発生してクラッシュする．

Fuzz4B の **ReproCrash** と GDB のブレークポイント機能を用いて，欠陥を含む行を特定する演習を行った．受講生は，欠陥を含む行を確認すると scanf 関数の呼び出しにおいて第 2 引数にアドレス演算子&が不足していることによりセグメンテーションフォールトが発生していることに気づく．最後に，第 2 引数にアドレス演算子を追加することで欠陥を修正するように指示した．

設問 1:
A．ファジングとデルタデバッギング，それぞれの役割を区別した上で，組み合わせたときの効果を説明できている．
B．ファジングとデルタデバッギング，それぞれの役割を区別できているが，組み合わせたときの効果を説明できていない．もしくは，組み合わせたときの効果を説明できているものの，両者の区別が曖昧である．
C．ファジングとデルタデバッギングの区別ができておらず，組み合わせたときの効果も説明できていない．
D．ファジングとデルタデバッギングについてどちらかに誤解があり，組み合わせたときの効果も説明できていない．
E．ファジングとデルタデバッギングについて両者に誤解があり，組み合わせたときの効果も説明できていない．

設問 2:
A．機能名について回答できていて，理由についても説明できている．
B．機能名について回答できていないが，理由についての説明ができている．
C．機能名について回答できているが，理由についての説明が不十分である．
D．機能名について回答できておらず，理由についての説明が不十分である．

設問 3:
A．GDB のみを使用してデバッグを行う場合と区別しながら，ファジングの結果を利用して GDB でデバッグする効果を説明できている．
B．GDB のみを使用してデバッグを行う場合とファジングの結果を利用して GDB でデバッグする場合の区別ができているが，ファジングの結果を利用して GDB でデバッグする効果についての説明が不十分である．
C．GDB のみを使用してデバッグを行う場合とファジングの結果を利用して GDB でデバッグする場合の区別が出てきていないが，ファジングの結果を利用して GDB でデバッグする効果についての説明はできている．
D．GDB のみを使用してデバッグを行う場合とファジングの結果を利用して GDB でデバッグする場合の区別ができておらず，ファジングの結果を利用して GDB でデバッグする効果についての説明も不十分である．

図 1　理解度確認テスト 2 の採点基準

3.6　レポート課題: Fuzz4B+GDB 演習

レポート課題として，演習 E と同様に **Fuzz4B** の **ReproCrash** を用いることでデバッグを行い，欠陥修正を行う演習を課した．対象プログラムは，ユーザが入力した整数値に対して，合計値や平均値の計算やソートをするプログラムである．演習 E の対象プログラムと同様に，数値の交換を行う swap 関数においてアドレス演算子が欠如しているため，セグメンテーションフォールトが発生する欠陥を含んでいる．

4　理解度確認テストやアンケートの結果

理解度確認テスト 2 では，以下の設問に回答してもらった．

設問 1　ファジングとデルタデバッギングを組み合

ファジングについて興味を持ちましたか	デルタデバッギングについて興味を持ちましたか	GDB等のデバッガについて興味を持ちましたか
4. まあまあ興味を持つことができた	4. まあまあ興味を持つことができた	4. まあまあ興味を持つことができた
4. まあまあ興味を持つことができた	4. まあまあ興味を持つことができた	4. まあまあ興味を持つことができた
4. まあまあ興味を持つことができた	4. まあまあ興味を持つことができた	4. まあまあ興味を持つことができた

ファジングについて理解できましたか	デルタデバッギングについて理解できましたか	GDB等のデバッガについて理解できましたか
5. 理解できた	5. 理解できた	5. 理解できた
2. あまり理解できなかった	2. あまり理解できなかった	2. あまり理解できなかった
4. まあまあ理解できた	4. まあまあ理解できた	4. まあまあ理解できた

図 2　理解度や興味に関するアンケート結果

表 1　理解度確認テスト 2 の採点結果

	設問 1	設問 2	設問 3
受講生 1	D	C	D
受講生 2	A	A	C
受講生 3	A	A	A

わせる効果について説明せよ．例えば，ファジングのみ使用した場合やデルタデバッギングのみ使用したときと比較しながら説明せよ．

設問 2　ファジングの結果を利用して GDB でデバッグすることを考える．GDB のどのような機能を使ってデバッグをするか答えよ．また，その機能を使用する理由を説明せよ．

設問 3　GDB のみを使用してデバッグを行う場合と，ファジングの結果を利用して GDB でデバッグする場合に，どのような違いがあるか説明せよ．

採点は，各設問ごとに図 1 に示す基準で行なった．A が最も高評価であり，E が最も低評価である．

理解度確認テスト 2 を提出した 3 名について，採点した結果を表 1 に示す．第一著者と第二著者が独立に採点を行ったところ，採点結果は完全に一致した．3 名の受講生の間で，理解度に大きな差があった．このことから，理解度の低い学生に対する対策が必要であることがわかる．なお，レポートについても理解度確認テスト 2 と同様に，受講生 3 名の間で成績に大きな差があった．

講義終了後に，ファジングやデルタデバッギング，デバッガに対する理解度と興味について，5 段階のリッカート尺度によるアンケートを実施した．アンケート結果を図 2 に示す．無記名アンケートであるため，表 1 の受講生との対応関係は不明である．このアンケート結果により，すべての受講生が講義であつかったファジングやデルタデバッギング，デバッガに興味を持つことはできたが，理解度に関しては受講生間で差異があることがわかる．アンケート結果の理解度に関する考察は，表 1 からの考察と一致している．

5　まとめと今後の課題

ファジングの活用事例を増加させることを目指し，ある大学にてファジングを題材とした講義を実施した．受講者のファジングに対する興味を促進することには成功したが，ファジングに対する理解度を促進することについては課題が残った．

また，理解度の向上を定量的に計測する方法を考案し，再度授業を実施することで定量的な評価を行うことも今後の課題である．

謝辞

本講義を実施するに当たって多大なご助力をいただいた，島根大学の神谷年洋教授に感謝いたします．本研究は，JST さきがけ JPMJPR21PA および JSPS 科研費 JP18H04094，JP19K14337 の支援を受けたものです．

参 考 文 献

[1] Böhme, M., Pham, V.-T., Nguyen, M.-D., and Roychoudhury, A.: Directed greybox fuzzing, *Proc. of CCS 2017*, pp. 2329–2344.

[2] 独立行政法人情報処理推進機構セキュリティセンター: ファジング活用の手引き, 独立行政法人情報処理推進機構, 2013.

[3] Garousi, V., Rainer, A., Lauvås, P., and Arcuri, A.: Software-testing education: A systematic liter-

ature mapping, *Journal of Systems and Software*, Vol. 165(2020), pp. 110570.

[4] 宮木龍, 吉田則裕, 都築夏樹, 山本椋太, 高田広章: Fuzz4B: ファジングツール AFL の利用支援ツール, 情報処理学会研究報告, Vol. 2020-SE-205, No. 6(2020), pp. 1–8.

[5] Miyaki, R., Yoshida, N., Tsuzuki, N., Yamamoto, R., and Takada, H.: Fuzz4B: A Front-End to AFL Not Only for Fuzzing Experts, *Proc. of A-TEST 2020*, pp. 17–20.

[6] 森孝夫, 本田晋也, 石田利永子, 山本雅基, 高田広章: コンソーシアム型共同研究を通じたソフトウェアテスト技術者の教育, 工学教育, Vol. 59, No. 2(2011), pp. 85–90.

[7] 中野隆司, 田中裕大, ダンティホンイエン: ソフトウェアのテスト工数・期間を削減するためのシステムテスト自動化技術, 東芝レビュー, Vol. 73, No. 3(2018), pp. 40–44.

[8] Peng, H., Shoshitaishvili, Y., and Payer, M.: T-Fuzz: fuzzing by program transformation, *Proc. of S&P 2018*, pp. 697–710.

[9] 高澤亮平, 坂本一憲, 鷲崎弘宜, 深澤良彰: テスト貢献度に基づくゲーミフィケーションを用いた教育用ソフトウェアテストツールの提案, 情報処理学会第 *75* 回全国大会講演論文集, Vol. 2013, No. 1, mar 2013, pp. 427–428.

[10] Wen, C., Wang, H., Li, Y., Qin, S., Liu, Y., Xu, Z., Chen, H., Xie, X., Pu, G., and Liu, T.: Memlock: Memory usage guided fuzzing, *Proc. of ICSE 2020*, pp. 765–777.

[11] Zeller, A. and Hildebrandt, R.: Simplifying and isolating failure-inducing input, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 2(2002), pp. 183–200.

[12] Zivkovic, T. and Zivkovic, M.: Survey of Learning Environments for Software Testing Education, *Proc. of the 7th Conference on the Engineering of Computer Based Systems*, ECBS 2021, New York, NY, USA, Association for Computing Machinery, 2021.