Środowiska i narzędzia związane z wytwarzaniem oprogramowania

Environment and tools related to software development

Michał Kuliś[1], Jarosław Jazienicki[2], Patryk Stachura[3]

Streszczenie: Niniejsza praca porusza problematykę i kwestie związane ze środowiskami, narzędziami oraz szeroko pojętym wytwarzaniem oprogramowania. Przedstawia również przykłady środowisk i narzędzi, które są na dzień dzisiejszy najczęściej wykorzystywane. Dzisiejsze tempo życia, wpływ globalizacji oraz bardzo szybki rozwój technologiczny budzą coraz to większe wymagania społeczeństwa w stosunku do jakości wytwarzanych usług, w tym także oprogramowań różnego rodzaju urządzeń. Zakres oferowanych środowisk oraz narzędzi, które mają za główne zadanie ułatwiać oraz zwiększać możliwości jest coraz szerszy i bardziej rozbudowany. Społeczeństwo wymaga coraz to bardziej zaawansowanych oraz bardziej skutecznych pod względem innowacyjności usług i rozwiązań problemów, które są związane z życiem codziennym. Wymusza to ogromną presję na koncernach, korporacjach, firmach, które zajmują się właśnie wytwarzaniem oprogramowania. Obawy te związane są ze skutecznością zaspokojenia popytu na dane usługi.

Abstract: This work addresses issues and issues related to environments, tools and the broadly understood software development. It also presents examples of environments and tools that are mostly used today. Today's pace of life, the impact of globalization and very rapid technological development are raising the society's ever-increasing demands in relation to the quality of services produced, including software of various types of devices. The range of offered environments and tools that have the main task to facilitate and increase the opportunities is growing wider and more extensive. Society requires more and more advanced and more effective in terms of innovation services and solutions to problems that are related to everyday life. This forces huge pressure on corporations, corporations, and companies that are involved in the production of software. These fears are related to the effectiveness of satisfying the demand for given services.

Słowa kluczowe: oprogramowanie, środowisko programistyczne, IDE, narzędzie wytwarzania oprogramowania, język programowania.

Keywords: software, development environment, IDE, software development tool, programming language.

Wstęp

Na samym początku warto się zaznajomić z podstawowymi pojęciami związanymi z wytwarzaniem programowania, takimi jak:

- Oprogramowanie,
- Proces wytwórczy oprogramowania,
- Narzędzie programistyczne,
- Zintegrowane środowisko programistyczne,
- Język programowania.

Oprogramowanie - całość informacji w postaci zestawu instrukcji, zaimplementowanych interfejsów i zintegrowanych danych przeznaczonych dla komputera do realizacji wyznaczonych celów. Celem oprogramowania jest przetwarzanie danych w określonym przez twórcę zakresie. Oprogramowanie tworzą programiści w procesie programowania. Oprogramowanie pisane jest zazwyczaj przy użyciu różnych języków programowania z wykorzystaniem algorytmów. Programy przekształcające oprogramowanie z postaci źródłowej na binarną to kompilatory. Niektóre oprogramowanie, np. napisane w całości w językach interpretowanych, może występować tylko w jednej postaci, spełniającej oba zadania [1].

Proces wytwórczy oprogramowania - proces mający na celu wytworzenie oprogramowania. Oprogramowanie wytwarzane jest od stosunkowo niedawna, dlatego procesy wytwórcze oprogramowania szybko się zmieniają w czasie, zmienia się też często opinia na temat jakości i efektywności poszczególnych procesów [2].

Narzędzie programistyczne - program komputerowy służący do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. Do narzędzi programistycznych należą: kompilatory, asemblery, debuggery i zintegrowane

1. Student, Wrocławska Wyższa Szkoła Informatyki Stosowanej, ul. Adres, mail michalkulis1993@gmail.com
2. Student, Wrocławska Wyższa Szkoła Informatyki Stosowanej, ul. Adres, mail jarek.jazienicki@gmail.com
3. Student, Wrocławska Wyższa Szkoła Informatyki Stosowanej, ul. Adres, mail stachura.patryk@yahoo.com

środowiska programistyczne [3].

Zintegrowane środowisko programistyczne, IDE - aplikacja lub zespół aplikacji (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania. Aplikacje będące zintegrowanymi środowiskami programistycznymi charakteryzują się tym, że udostępniają złożoną, wieloraką funkcjonalność obejmującą edycję kodu źródłowego, kompilowanie kodu źródłowego, tworzenie zasobów programu, tworzenie baz danych, komponentów i innych [4].

Język programowania - zbiór zasad określających, kiedy ciąg symboli tworzy program komputerowy oraz jakie obliczenia opisuje [5].

Na wstępie warto również zaznaczyć, iż w dobie dzisiejszego tempa rozwoju technologii wszelakie środowiska oraz narzędzie programistyczne ulegają diametralnej zmianie każdego dnia. Codziennie powstają nowe frameworki, mające na celu wspomaganie danych komponentów języków programowania, nowe narzędzia oraz nowe środowiska. Same języki programowania nie ulegają tak szybkiej zmianie, lecz są równie szybko rozbudowywane o nowe funkcjonalności i frameworki, o których wspomnieliśmy powyżej. Także śmiało możemy pokusić się o tezę, iż również proces wytwarzania ulega zmianom, ponieważ jest znacząco uzależniony od komponentów, które się na niego składają. Proces tworzenia oprogramowania składa się z poszczególnych etapów, jest to także uzależnione od modelu procesu, który się zaimplementuje w projekcie.

Problematyka wytwarzania oprogramowania

Współczesne metodyki wytwarzania oprogramowania wykorzystują wiele artefaktów takich jak: raporty o błędach, kod źródłowy, harmonogramy, dokumentacja czy testy. Zintegrowane środowiska oraz narzędzia wspomagają zarządzanie i tworzenie tych artefaktów. Bardzo często pochodzą one od innych producentów dlatego różnią się one zarówno zakresem pokrywania procesu produkcji oprogramowania jak i funkcjonalnością, które oferują Zarówno narzędzia jak i środowiska te są wyspecjalizowane pod względem funkcji jakie oferują. Ich główne przeznaczenie to m.in.: zarządzanie wymaganiami, tworzenie dokumentacji, modelowanie oprogramowania, wytwarzanie dokumentacji itp. Bardzo rzadkim zjawiskiem jest sytuacja, gdzie producent w swojej ofercie posiada produkty, które pokrywają cały proces wytwarzania oprogramowania wraz z możliwością komunikowania się oraz wymiany danych, aby ułatwić ten proces programistyczny.

W dzisiejszych czasach palącym problemem inżynierii oprogramowania jest utrzymywanie spójności oraz zgodności pomiędzy artefaktami projektu. Za przykład może nam tu posłużyć pamiętanie o odzwierciedlaniu zmian w kodzie źródłowym oraz w przedstawiających go diagramach UML, co jest koszmarem dla większości programistów. Wyjściem z tej sytuacji jest wsparcie tej czynności za pomocą narzędzi CASE, które wyposażone są w możliwość reinżynierii danego oprogramowania i integrację ze środowiskiem programistycznym. Omawiany problem występuje również przy takich procesach jak: aktualizacja dokumentacji, harmonogramów, testów oraz innych elementów projektu. Jest to trudność ogólna dotycząca reagowania na zmiany oraz propagowania tych zmian w projekcie, w taki sposób aby zachować spójność jego elementów. Istotny jest fakt, iż koszt który jest wymagany przez dokładną analizę zależności tzn. Formalny proces propagacji zmian, przeważnie jest większy niż potencjalne zyski. W związku z tym zauważalne jest, iż bardzo duży odsetek projektów informatycznych rezygnuje z wszelakiej dokumentacji i modeli UML, w momencie przekroczenia pewnego stopnia skomplikowania projektu. W myśl integracji narzędzi programistycznych oraz środowisk zauważalna jest częściowa możliwość zautomatyzowania większości nużących i pracochłonnych czynności, które są związane z propagacją zmian, co prowadzi do poprawienia jakości projektu. Aby rozwiązać powyższy problem integracji środowisk i narzędzi mamy możliwe do zastosowania dwie opcje. Pierwszą z opcji jest wyprodukowanie lub powiązanie ze sobą takiego zestawu narzędzi, który da możliwość zbudowania rozproszonego I ujednoliconego środowiska zarządzającego wszystkimi elementami danego projektu oraz zintegrowania ich ze sobą. Poprzez rozproszenie rozumiane są tutaj zarówno geograficzne rozproszenie wykonawców projektu jak I zarówno komponentów system. Natomiast drugą możliwością jest próba wytworzenia szkieletowej architektury, co dało by możliwość zestawienia istniejących narzędzi a także stworzenia z nich jednej spójnej platform, która służyła by to wytwarzania oprogramowania [6].

Podstawowe procesy wytwarzania oprogramowania

Jak już wspominaliśmy we wstępie proces wytwarzania oprogramowania bardzo silnie opiera się o model procesu wytwarzania. Do podstawowych modeli wytwarzania możemy zaliczyć m.in.:

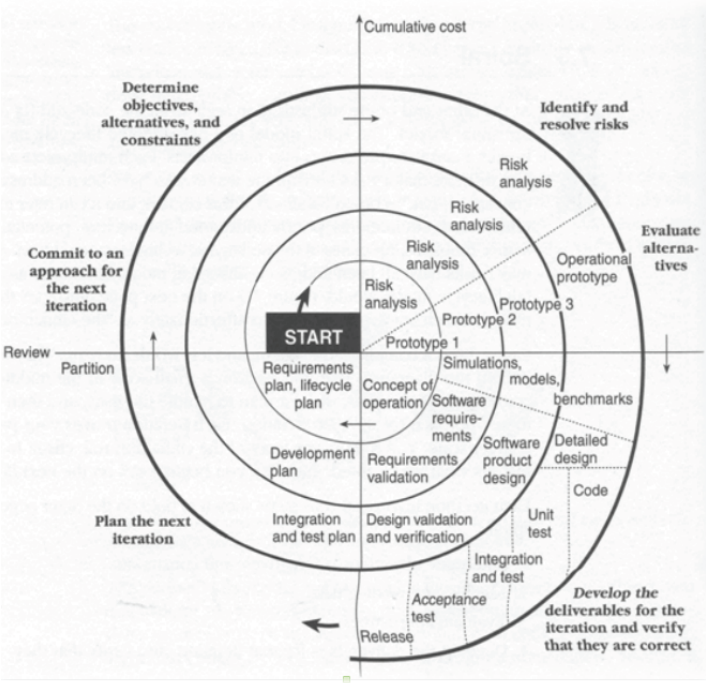
- Model kaskadowy,
- Prototypowanie,
- Model Spiralny,
- Model przyrostowy,
- Model przyrostowo-iteracyjny,
- Programowanie ekstremalne,
- Scrum (młyn) [7].
-

Model kaskadowy – model charakteryzuje sześć etapów realizacji. Pierwszym etapem jest proces planowania, określa się specyfikację wymagań projektu. Kolejny etap to analiza pod względem wykonalności projektu oraz analiza oferty. Trzecim etapem jest projektowanie działań oraz elementów projektu. Następnie następuje implementacja. Po zaimplementowaniu następuje etap testowania projektu, poszczególnych modułów oraz integracji. Końcowym

etapem jest wdrożenie i konserwacja. Wadami tego modelu może być brak elastyczności.

Prototypowanie – ten model głównie składa się z trzech etapów. Pierwszym etapem jest stworzenie prototypu. Następnie następuje szereg konsultacji z klientem, po czym ostatnim etapem jest realizacja projektu. Zaletą tego modelu jest mniejszy koszt wprowadzania jakichkolwiek zmian w prototypie. Natomiast główna wada to ogromny koszt realizacji systemu.

Model spiralny – model spiralny łączy w sobie cechy modelu kaskadowego oraz prototypowego. Kolejne prototypy tworzone są podobnie jak w modelu kaskadowym. Natomiast końcowym etapem jest stworzenie ostatecznej wersji produktu.

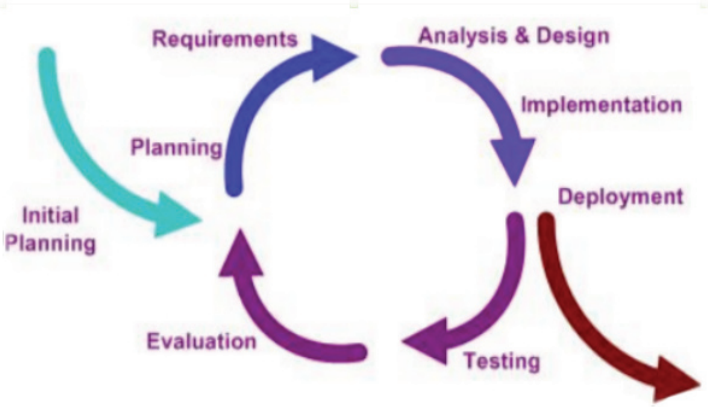


Ryc. 1. Model spiralny.

Model przyrostowy – model ten bazuje głównie na modelu kaskadowym. Proces wytwarzania oprogramowania jest realizowany w ten sposób, iż wraz z implementacją kolejnego etapu dodawana jest nowa funkcjonalność. Główne zalety tego modelu to łatwość dostosowywania modelu do zmian w specyfikacji oraz fakt, iż po zakończeniu każdego etapu mamy część zrealizowanego produktu.

Model przyrostowo-iteracyjny – model ten wykonywany jest w ramach kolejnych etapów modelu przyrostowego, jest to jego rozszerzenie. Model przyrostowo-iteracyjny jest jednym z najczęściej wykorzystywanych modeli, które służą do wytwarzania oprogramowania. Jeżeli chodzi o najpopularniejsze implementacje tego modelu:

- Rational Unified Process,
- Agile Software Development.



Ryc. 2. Model przyrostowo-iteracyjny.

Programowanie ekstremalne – model ten charakteryzuje głównie iteracyjność, co oznacza aktualizowanie produktu o nowe wersje co kilka tygodni. Występuję tutaj również brak dokumentacji oraz dopuszczalne są zmiany architektury projektu. Istotny w tym modelu jest stały kontakt z klientem. Testy jednostkowe tworzone są przed oprogramowaniem.

Scrum (młyn) – model scrum charakteryzuje się dobraniem samoorganizującego się oraz interdyscyplinarnego zespołu osób. Cykle, czyli sprinty trwają około 2-6 tygodni, efektem każdego jest działający produkt. Występują tutaj codzienne "poranne" kilkuminutowe spotkania teamu, w celu omówienia wczorajszych zadań, postępów i aktualnych celów [7].

Przykładowe środowiska, narzędzia wytwarzania oprogramowania

Jeżeli chodzi o narzędzia i środowiska programistyczne to możemy tu wyróżnić zintegrowane środowiska programistyczne(IDE). Są one wyposażone w szereg zaawansowanych menadżerów, które pokazują różne informacje o strukturze aplikacji oraz posiadają systemy wykrywania i usuwania błędów. W dzisiejszych czasach mamy do dyspozycji ich szeroki wachlarz. Zależnie od oczekiwanych usług i możliwości mamy do dyspozycji środowiska różnego przeznaczenia oraz służące do różnych celów. Czasami narzędzia programistyczne są tak wspierane oraz rozwijane, że z czasem stają się zintegrowanymi środowiskami programistycznymi. Do przykładowych środowisk programistycznych możemy zaliczyć m.in.:

- **Microsoft Visual Studio** – jest to zintegrowane środowisko programistyczne firmy Microsoft. Jest przeznaczone głównie do tworzenia oprogramowania konsolowego ale i również do tworzenia graficznych interfejsów użytkownika, w tym aplikacje WPF, Windows Forms, Web Aplications i innych. Produkty mogą być tworzone na platformy: Microsoft Windows, .NET Framework, XBOX, Microsoft Silverlight. Główne narzędzia tego środowiska to: Microsoft Visual C#, Microsoft Visual C++, Microsoft Visual Basic, Microsoft Visual J#, Mi-

crosoft Visual Web Developer ASP.NET oraz Microsoft Visual F# [8].

- **Eclipse** – jest to zintegrowane środowisko programistyczne stworzone do realizacji projektów typu rich-client. Głównym przeznaczeniem tego środowiska jest tworzenie aplikacji w języku programowania Java. Platforma została stworzona przez IBM w 2004r. Eclipse posiada szereg różnego rodzaju wtyczek, które umożliwiają głównie tworzenie produktów w takich językach jak: Java, C/C++, PHP a także tworzenie GUI, modelowanie aplikacji za pomocą UML i współpracę z bazami danych czy serwerami aplikacji [9].
- **.NET** – platforma programistyczna stworzona przez firmę Microsoft. To zintegrowane środowisko programistyczne nie jest związane z żadną konkretnie technologią, natomiast daje możliwość tworzenia produktów w wielu językach, np. C++/CLI, J#, C#, F#, Visual Basic czy Delphi. To środowisko daje możliwość tworzenia aplikacji działających po stronie serwera internetowego oraz aplikacji, które działają na różnych systemach. Główne narzędzia platformy .NET to przede wszystkim ASP.NET, które ułatwia tworzenie dynamicznych stron www oraz ADO.NET, które ułatwia dostęp oraz współpracę z bazami danych [10].

Natomiast typowe narzędzia, które wspierają pod względem upłynnienia i automatyzacji proces wytwarzania oprogramowania, mamy tutaj głównie na myśli system kontroli wersji. Wiele komercyjnych środowisk programistycznych jest zintegrowanych z **systemem kontroli wersji**. System kontroli wersji głównie służy do nadzorowania, zarządzania i integrowania wspólnego kodu. Przykłady takich narzędzi to m.in.:

- CVS
- Subversion
- GIT
- Bazaar itp [11].

Bardzo ważną grupą są narzędzia dające możliwość na usuwanie błędów z gotowej już aplikacji, produktu, czyli tzw. proces **debugowanie**. Możemy wyróżnić dwa różne sposoby debugowania:

- Statyczna analiza kodu,
- Dynamiczna analiza kodu.

Pierwszy sposób polega na analizie kodu pod względem możliwości wystąpienia błędów, natomiast drugi sposób polega na analizie kodu podczas wykonywania pracy produktu. Służą do tego odpowiednie narzędzia, zwane **debuggerami**. Istnieją również specjalne fragment kodu, które nastawione są na weryfikowanie kodu wyszukując w nim błędów. Niektóre języki mają wbudowane w swoje struktury debuggery i kod debugowany jest automatycznie, natomiast języki niskopoziomowe jak C czy C++ nie posiadają wbudowanych narzędzi debugujących [12].

Kolejna grupa narzędzi niezbędnych przy tworzeniu oprogramowania to **kompilatory**. Kompilator jest to narzędzie odpowiedzialne za przetłumaczenie(skompilowanie) języka programowania na język maszynowy, zrozumiały dla komputera. Niektóre kompilatory najpierw tłumaczą kod programistyczny na kod assemblera a ten kolejno na język maszynowy. Zaletą kompilatorów jest to, iż programista nie musi znać języka maszynowego oraz daje możliwość sporej przenośności danych pomiędzy platformami. Jako przykłady kompilatorów możemy wymienić m.in.:

- GNU Fortran 77[FORTRAN],
- GNU C[C],
- GNU C++[C++],
- Sun Java Compiler, GNU Java Compiler[JAVA] itp [13].

Istotnymi narzędziami jakie wprowadzają ogromną innowacyjność w proces wytwarzania oprogramowania są **biblioteki** oraz **frameworki** języków programowania. Biblioteki wraz z frameworkami są tworzone bardzo dynamicznie oraz często. Sam fakt o ilości bibliotek oraz frameworków biorąc pod uwagę wszystkie języki programowania jest nieograniczenie potężna z racji tego, iż są one wytwarzane niemal cały czas. Przykładowe biblioteki czy frameworki to:

- Symfony[PHP],
- React[JS],
- Angular[JS],
- Spring[JAVA],
- Entity[C#] itp.

Natomiast najważniejszy element przy wytwarzaniu oprogramowania to sam **język programowania**. To właśnie za pomocą języka programowania wytwarzane jest oprogramowanie. Języki programowania dzieli się potocznie na języki:

- Niskiego poziomu,
- Wysokiego poziomu.

Do języków niskiego poziomu zalicza się głównie:

- Język maszynowy,
- Assembler.

Jeżeli chodzi o języki wysokiego poziomu, to są to m.in. :

- Java,
- C#,
- C/C++,
- Python,
- Pascal,
- PHP itp. [14].

Kolejne narzędzie, o którym należy wspomnieć jest **Unified Modeling Language.** UML jest tak naprawdę zunifikowanym językiem modelowania. Przede wszystkim pozwala tworzyć różnego rodzaju modele(np. informatyczne). Pozwala obrazować, tworzyć, specyfikować oraz dokumentować element tworzonego system. UML ułatwia także wykorzystanie zalet, które występują wraz z programowaniem obiektowym oraz jest przeznaczony głównie do używania w procesie analizy i projektowania systemów komputerowych. UML jest złożony z dwóch elementarnych składowych:

- Notacja,
- Metamodel [15].

Notacja	Metamodel
• element graficzne • składnia języka modelowania • istota przy szkicowaniu	• definicje pojęć języka i powiązania między nimi • istotny przy graficznym programowaniu

Tab. 1. Składowe UML.

Unified Modeling Language składa się na dwie powyższe definicje. Pierwsza czyli notacja wyznaczonych elementów, które są używane na diagramach, natomiast z drugiej strony ich semantykę, czyli metamodel. Najważniejsze jest to aby model był jednoznacznie i czytelnie opisany, tak aby inne osoby bez problem potrafiły go zrozumieć. Dlatego kluczowa jest tutaj notacja, zaś metamodel winien być zrozumiały intuicyjnie. Przy generowaniu kodu oraz przejściu do implementacji tego kodu, najważniejsze jest ścisłe zrozumienie przeznaczenia określonych elementów, aby możliwa była automatyczna konwersja modelu do innego formalizmu.
Zakres oraz przeznaczenie zastosowania UML zawiera się głównie w:

- Tworzeniu systemów informacyjnych przedsiębiorstw,
- Usługach bankowych I finansowych,
- Transporcie,
- Telekomunikacji,
- Sprzedaży detalicznej,
- Przemyśle obronnym i lotniczym,
- Nauce,
- Elektronice i medycynie,
- Rozproszonych usługach internetowych [15].

UML tworząc określony model bazuje na odpowiednich diagramach, rodzaje tych diagramów to m.in. [15]:

Diagramy strukturalne	Diagramy dynamiki
• Diagram klas • Diagram obiektów • Diagram pakietów • Diagram komponentów • Diagram wdrożenia • Diagram struktur złożonych	• Diagram przypadków użycia • Diagram stanów • Diagram przebiegu • Diagram czynności • Diagram kooperacji • Diagram przeglądu interakcji • Diagram uwarunkowań czasowych

Tab. 2. Rodzaje diagramów UML.

Narzędzia **CASE** są również istotne w procesie tworzenia oprogramowania. Computer Aided Software Engineering jest to oprogramowanie używane do komputerowego wspomagania projektowania oprogramowania. Główne funkcje CASE to: analiza, projektowanie oraz programowanie. Narzędzia CASE automatyzują metody dokumentacji, projektowania oraz tworzenia struktur kodu program w określonym języku programowania obiektowego. Głównymi narzędziami CASE są:

- Narzędzia do modelowania w UML oraz podobnych,
- Narzędzia do zarządzania systemem kontroli wersji,
- Narzędzia do refactoringu, czyli do wprowadzania zmian w danym projekcie [16].

Wnioski

Już dzisiaj widzimy w jak szybkim tempie postępuje rozwój technologiczny oraz jakie ogromne możliwości daje nam rozwój komputerów i internetu. Postęp ten spowodował, iż granice jakimi były czas oraz odległość wraz z wpływem globalizacji i nowych technologii zamazały się. Bez rozwoju stopnia zaawansowania szeroko pojętego oprogramowania na dzień dzisiejszy rozwój na tak wysokim poziomie nie był by możliwy. Natomiast tempo rozwoju technologicznego ciągle wzrasta, co za tym idzie dostępność oraz ilość narzędzi i środowisk wysoko zaawansowanych także rośnie, co pozytywnie wpływa na automatyzacje i innowacyjność procesu wytwarzania oprogramowania ale nie tylko. Wzrost poziomu zaawansowania w procesie wytwarzania oprogramowania znacząco wpływa na wzrost komfortu życia społeczeństwa oraz jego funkcjonowania. Możemy śmiało pokusić się o tezę, iż urządzenia elektroniczne, co za tym idzie oprogramowanie jest na dzień dzisiejszy wszechobecne i odgrywa ogromne znaczenie na funkcjonowanie w życiu codziennym. Jeżeli chodzi o kierunek rozwoju tworzenia oprogramowań, to rozwój zmierza ku stworzeniu sztucznej inteligencji, co w najbliższej przyszłości będzie możliwe. Już możemy zauważyć prototypy autonomicznych maszyn, robotów oraz urządzeń wspomagających w życiu codziennym społeczeństwo.

Literatura

[1] http://web.archive.org/web/20010530092843/http://home.olemiss.edu/~misbook/sfsysfm.htm, dostęp: 15.04.2018r.
[2] http://edu.pjwstk.edu.pl/wyklady/pri/scb/index05.html, dostęp: 15.04.2018r.
[3] Kernighan, Brian W.; Plauger, P. J. (1976), Software Tools, Addison-Wesley, p. 352.
[4] https://pl.wikibooks.org/wiki/C/Zintegrowane_%C5%9Brodowisko_programistyczne, dostęp: 15.04.2018r.
[5] Mordechai Ben-Ari: Understanding Programming Languages. Chichester: John Wiley & Sons, 1996r.
[6] C. Boldyreff i R. Dewar, Environments to Support Collaborative Software Engineering., 2003, 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes, Benevento.
[7] Górski J. „Inżynieria Oprogramowania w projekcie informatycznym", Mikom, s. 74-84.
[8] https://docs.microsoft.com/pl-pl/visualstudio/releasenotes/vs2017-relnotes, dostęp: 15.04.2018r.

[9] http://help.eclipse.org/oxygen/index.jsp, dostęp: 15.04.2018r.
[10] Microsoft makes .Net open-source, finally embraces iOS, Android, and Linux - ExtremeTech, www.extremetech.com, dostęp: 15.04.2018r.
[11] https://docs.microsoft.com/pl-pl/vsts/tfvc/overview?view=vsts, dostęp: 15.04.2018r.
[12] Andreas Zeller: Why Programs Fail: A Guide to Systematic Debugging. Morgan Kaufmann, 2005.
[13] A.V. Aho, R. Sethi, J.D. Ullman, Kompilatory. Reguły, metody i narzędzia, WNT, Warszawa 2002.
[14] Bruce J. MacLennan: Principles of Programming Languages. Oxford University Press, 1987,s. 132.
[15] Tom Pender: UML Bible. John Wiley & Sons, 2003.
[16] A. Jaszkiewicz, Inżynieria oprogramowania, Helion, Gliwice, 1997, s. 112.
[17] P.Kardasz, Programowanie w języku Python, Wrocław 2017