PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr 1134 — 2006
Nowoczesne technologie informacyjne w zarządzaniu

Andrzej Kobyliński, Tomasz Chodoła

ZŁOŻONOŚĆ OPROGRAMOWANIA – INTERPRETACJE POJĘCIA

1. Istota złożoności

Złożoność oprogramowania jest powszechnie uznawana za jedną z głównych przyczyn problemów, na jakie napotykają twórcy oprogramowania. Efektem tego jest powszechne przekraczanie budżetów, niedotrzymywanie terminów i niezadowalająca jakość oddawanych produktów. Brooks [2000, s. 159-183] zaliczył złożoność, obok zgodności, zmienności i niewidoczności, do przyczyn zasadniczych, niemożliwych do zwalczenia nie tylko obecnie, ale i w przyszłości. Złożoność, zgodnie ze słownikową definicją, rozumiana jest jako „to, że się coś składa z wielu elementów, to, że coś ma budowę skomplikowaną" [*Słownik języka*... 1989, s. 1033]. Złożoność oprogramowania stwarza o wiele większe problemy, niż złożoność, z którą mamy do czynienia w przypadku produktów materialnych [Kobyliński 2006]. Konsekwencje złożoności oprogramowania są bardzo poważne. W szczególności wszystkie metody służące do wymiarowania (np. punkty funkcyjne) lub estymacji oprogramowania (np. COCOMO) wymagają apriorycznej oceny złożoności składowych realizowanego przedsięwzięcia. Niewłaściwe oszacowanie złożoności wpływa w konsekwencji na ustalenie nierealnych planów dotyczących budżetu i harmonogramu i skutkuje ostatecznie niemożnością osiągnięcia założonych celów.

W literaturze dotyczącej inżynierii oprogramowania można spotkać się z bardzo różnym podejściem do złożoności. Autorzy posługujący się tym terminem rzadko określają, jaki rodzaj złożoności mają na myśli. Tymczasem sposobów interpretacji tej abstrakcyjnej kategorii jest bardzo wiele. Celem niniejszej pracy jest dokonanie ich przeglądu. Świadomość wieloznaczności tego terminu może się przyczynić w dalszej perspektywie do większego zdyscyplinowania w jego stosowaniu.

2. Przykłady interpretacji pojęcia „złożoność" w odniesieniu do oprogramowania

W literaturze naukowej i technicznej znaleźć można odniesienia do wielu różnych rodzajów złożoności. Poniżej przedstawiono niektóre z nich – te, które mogą być zastosowane w kontekście oprogramowania.

Złożonością psychologiczną (*psychological complexity*) programu nazywa się stopień trudności jego zrozumienia, gdy programista stara się pojąć cel i funkcję programu oraz sposób jego realizacji, przy uwzględnieniu trzech elementów: czynnika ludzkiego, złożoności kodu i dostępnego materiału informacyjnego innego niż sam program. Uwzględnia się tutaj subiektywny pogląd personelu programującego i konserwującego na temat tego, czy kod, za który są odpowiedzialni, jest złożony, czy też nie. Ten rodzaj złożoności uwzględnia oprócz elementu wyliczanego obiektywnie (złożoności kodu) również zdolności i doświadczenie osób pracujących nad programem, a także np. dostępną dokumentację [Ndayambaye 1990].

Złożonością kodu (*code complexity*) nazywany jest stopień trudności zrozumienia działania programu, określany wyłącznie na podstawie jego kodu źródłowego [Kobyliński 1991]. Jest to pojęcie węższe od omówionej wyżej złożoności psychologicznej, gdyż uwzględnia wyłącznie te własności programu, które są dla tego programu stałe i niezmienne – nie zależy w szczególności od umiejętności personelu opracowującego go i konserwującego. Zebranie obiektywnych informacji na temat złożoności kodu jest możliwe, jeśli użyje się do tego obiektywnych, wyliczanych automatycznie miar (wielkości, struktury, danych itd.).

Złożoność semantyczna (znaczeniowa) (*semantic complexity*) wywodzi się z lingwistyki i dotyczy niejednoznaczności używanych terminów [Pollard, Biermann 2000]. Pojęcie złożoności semantycznej jest istotne w odniesieniu do oprogramowania z pewnego zaskakującego powodu: wiele procesów sądowych między wytwórcami oprogramowania a ich klientami wynika z semantycznej złożoności kontraktu, kiedy obie strony przyjmują odmienną interpretację tych samych pojęć.

Złożoność syntaktyczna (składniowa) (*syntactic complexity*) również pochodzi z lingwistyki i odnosi się do długości podstawowych struktur tekstu (takich jak akapity, zdania lub słowa) i ich struktury gramatycznej [Flank 2000]. Pomiar złożoności syntaktycznej jest stosowany od kilkudziesięciu lat, a najpopularniejsze jego miary to indeksy: FOG [Gunning 1952], Dale–Challa [Chall, Dale 1995], Flesch–Kincaida [Flesch 1979] i Fry'a [Fry 1977]. Wydaje się, że indeksy tego rodzaju mogą być używane w odniesieniu do specyfikacji oprogramowania zapisanej w języku naturalnym, a mimo że niektóre edytory tekstów mają wbudowaną funkcję ich obliczania, to praktycznie możliwości te nigdy nie są wykorzystywane.

Złożoność obliczeniowa (*computational complexity*) odnosi się do maksymalnej liczby iteracji (lub jednostek czasu maszynowego) albo ilości pamięci komputerowej potrzebnych do rozwiązania konkretnego problemu zgodnie z pewnym algorytmem [Błażewicz 1988]. Niektóre problemy mają tak wysoką złożoność

obliczeniową, że są uważane za nie do zrealizowania. Inne problemy są realizowalne, ale wymagają ogromnej ilości czasu maszynowego (np. kryptoanaliza lub analiza wzorców pogodowych). Złożoność obliczeniowa może być użyta do oceny wydajności programu, ale nie trudności podczas jej budowy lub konserwacji (proste, łatwe do zaimplementowania algorytmy często mają dużą złożoność obliczeniową, a zwiększając złożoność algorytmu, udaje się niekiedy zmniejszyć złożoność obliczeniową).

Złożoność algorytmiczna (*algorithmic complexity*) uwzględnia długość i strukturę algorytmu problemu obliczeniowego. O tym, jak złożony jest system, wnioskuje się na podstawie najkrótszego programu komputerowego albo zbioru algorytmów, wymaganych do kompletnego opisu systemu[1]. Aplikacje oparte na długich i skomplikowanych pojęciowo algorytmach są trudne do projektowania, kodowania, dowodzenia, testowania i uruchamiania, co wpływa na jakość produktu i produktywność pracowników.

Złożoność problemu (*problem complexity*) w odniesieniu do zachowania się pewnego systemu określana jest przez nieznane elementy, takie jak składniki, relacje, siły, powiązania, ograniczenia systemu, jakie badacz problemu musi ustalić, aby rozwiązać problem [Waxman 1996]. Złożoność problemu jest sprawą subiektywną. Różne osoby posiadające różne wizje systemu, więc posiadające odmienne braki w wiedzy o systemie, będą miały odmienny pogląd na temat złożoności problemu do rozwiązania. Kiedy o systemie wiadomo już wszystko, co jest potrzebne, to problem związany z systemem przestaje być złożony, chociaż rozwiązanie problemu może być nadal bardzo skomplikowane i wymagać olbrzymiej ilości pracy.

Złożoność organizacyjna (*organizational complexity*) dotyczy sposobu, w jaki zorganizowana jest praca w firmie wytwarzającej oprogramowanie [*Managing Organizational...* 2005]. Wpływ złożoności organizacyjnej na wytworzony produkt programowy jest dlatego istotny, że niektóre duże projekty informatyczne są dzielone na części w taki sposób, że odzwierciedlają aktualną strukturę organizacji wytwórczej, a nie techniczne potrzeby samego projektu informatycznego. Wiadomo, że na ogół duże projekty informatyczne, realizowane w firmach o dobrym stopniu zorganizowania (oceniane na wysoki poziom CMMI), przewyższają podobne projekty realizowane w słabo zorganizowanych strukturach.

Złożoność procesu (*process complexity*) w pracach związanych z wytwarzaniem oprogramowania odnosi się do przepływu pracy i dokumentów wytworzonych w całym cyklu życia oprogramowania. Niestety, do tej pory nie zaproponowano metody, która by umożliwiła analizowanie i redukcję takiej złożoności [Cardoso 2005]. Ten rodzaj złożoności jest często praktycznie opanowywany dzięki narzędziom wspomagającym zarządzanie przedsięwzięciem, które to narzędzia

[1] Pojęcie złożoności algorytmicznej zaproponowane zostało przez Chaitina, który oparł się na pracach Shannona, Kolmogorowa i Solomonowa [*Encyclopedia of Astrobiology...* 2006].

pozwalają określić ścieżkę krytyczną oraz diagramy PERT (*program evaluation and review technique*) procesu budowy oprogramowania.

Złożoność entropiczna (*entropic complexity*) odzwierciedla stan nieuporządkowania składowych części systemu. Zjawisko entropii, początkowo zaobserwowane w termodynamice, w późniejszym okresie zostało odnotowane również w innych obszarach. Entropia jest o tyle ważnym pojęciem, że wszystkie znane systemy wykazują wzrost entropii w czasie, co oznacza, że nieuporządkowanie stopniowo się zwiększa. Zjawisko to zauważono również w projektach informatycznych, ponieważ wiele małych zmian w czasie stopniowo powoduje erozję (degraduje) oryginalną strukturę i jest uważane za jedną z podstawowych przyczyn „eksplozji systemu" i wycofywania długo wykorzystywanego oprogramowania z eksploatacji. Długotrwałe studia nad projektami informatycznymi przebywającymi w fazie konserwacji próbują mierzyć stopień, w jakim entropia się zwiększyła, i określić, czy może ona być ponownie zmniejszona poprzez takie działania, jak restrukturyzacja kodu. Wprawdzie nie istnieją bezpośrednie miary entropii oprogramowania, ale zastępczymi miarami służącymi do oceny złożoności entropicznej są współczynniki oceniające, jak zmieniła się złożoność cyklomatyczna lub esencjonalna w czasie, np. w ciągu roku.

Złożoność według „teorii oprogramowania" (*software science complexity*) wywodzi się z badań lingwistycznych nad programami komputerowymi, prowadzonych przez Halsteada w końcu lat siedemdziesiątych [Halstead 1977]. Teoria oprogramowania traktuje program komputerowy jako ciąg symboli, które można zaliczyć do jednej z dwóch grup: operatorów i operandów (argumentów). Według Halsteada, złożoność można policzyć opierając się na czterech miarach podstawowych:

1) liczbie unikalnych operatorów,
2) liczbie unikalnych operandów,
3) ogólnej liczbie operatorów w programie,
4) ogólnej liczbie operandów w programie.

Halstead wyróżnia w swej teorii takie pojęcia, jak swoiście rozumiane: słownik programu, długość programu, objętość programu. Uzasadnia, w jaki sposób można prognozować te wartości jeszcze przed przystąpieniem do kodowania. Umożliwia to stosunkowo wczesne estymacje wysiłku i czasu, jakie będą wymagane do napisania programu. Co do miar złożoności Halsteada, aczkolwiek literatura na ich temat jest dość bogata, pojawiło się wiele wątpliwości i krytycznych uwag [Shen, Conte, Dunsmore 1983].

Złożoność funkcjonalna (*function point complexity*) odnosi się do zbioru parametrów podstawowych, kwantytatywnych, służących do obliczania rozmiaru funkcjonalnego badanego systemu oraz czynników korygujących, potrzebnych do obliczenia ostatecznej liczby punktów funkcyjnych przedsięwzięcia informatycznego [Czarnacka-Chrobot 2001]. Każdy z pięciu parametrów podstawowych oceniany

jest na jeden z trzech wyróżnianych poziomów złożoności. Mimo wytycznych, jakie co kilka lat publikuje IFPUG (International Function Point Users Group – Międzynarodowa Grupa Użytkowników Punktów Funkcyjnych), obiektywizm tej oceny budzi ciągłe wątpliwości. Również wśród parametrów wpływu (zwanych też czynnikami korygującymi), jakich jest 14 w standardowej amerykańskiej definicji IFPUG, znajdują się dwa, które wymagają subiektywnej oceny złożoności:

1) wejścia, wyjścia i zbiory danych,
2) wewnętrzne przetwarzanie danych w systemie.

Złożoność poszczególnych parametrów oraz wartości czynników korygujących jest zazwyczaj oceniana na zasadzie odwołań do tablicy znanych wartości, niemniej obarczona jest dużą dozą subiektywizmu. Przez to nie jest możliwe automatyczne obliczanie funkcjonalności produktu programowego za pomocą punktów funkcyjnych, nawet dla już istniejącego, gotowego, działającego produktu.

Złożoność grafu (*graph complexity*) wywodzi się z teorii grafów i można ją mierzyć na parę różnych sposobów, np. jako liczbę drzew rozpinających grafu albo, częściej, jako pewne wyrażenie, obliczone na podstawie liczby węzłów, krawędzi i ewentualnie innych cech grafu [Neel, Orrison 2006]. Koncepcja złożoności grafu jest ważna w odniesieniu do oprogramowania, gdyż na niej opiera się analiza złożoności cyklomatycznej i esencjonalnej. W przeszłości powstało również kilka innych miar złożoności struktury sterowania w programie, opierających się na złożoności grafu, ale zdecydowanie najpopularniejszą z nich jest miara złożoności cyklomatycznej.

Złożoność cyklomatyczna (*cyclomatic complexity*) wywodzi się z teorii grafów i została zaproponowana przez McCabe'a [McCabe 1976]. Złożoność cyklomatyczna jest miarą możliwej różnorodności przepływu sterowania w grafie obrazującym strukturę fragmentu oprogramowania. Można ją obliczać na cztery różne sposoby: jako liczbę skierowanych cykli elementarnych, regionów, ścieżek (dróg elementarnych) i warunków w tekście programu [Kobyliński 1991, s. 36-43]. Ostatni z wymienionych sposobów jest najpopularniejszy – ze względu na łatwość dokonywania obliczeń nie wymagających rysowania grafu – a opiera się wyłącznie na analizie leksykalnej kodu. Programy bez rozgałęzień mają złożoność cyklomatyczną równą 1; kiedy liczba rozgałęzień rośnie, zwiększa się też złożoność cyklomatyczna. Dla złożoności cyklomatycznej powyżej 20 testowanie staje się trudne, a dla większych wartości nawet niemożliwe. Złożoność cyklomatyczna bywa często stosowana jako wskaźnik ostrzegający przed potencjalnymi problemami jakościowymi. Złożoność cyklomatyczna jest najpopularniejszą formą analizy złożoności dla przedsięwzięć informatycznych i posiada obszerną literaturę.

Złożoność esencjonalna (*essential complexity*) również wywodzi się z teorii grafów i stanowi pewien wariant złożoności cyklomatycznej. Do obliczenia złożoności esencjonalnej niezbędne jest utworzenie grafu sterowania programu (tak jak jest to też wymagane przy trzech wcześniej wspomnianych sposobach obliczania

złożoności cyklomatycznej), a następnie usunięcia z grafu wszystkich podstawowych konstrukcji promowanych przez programowanie strukturalne (sekwencji, warunków, pętli) [Watson, McCabe 1996]. Uproszczony w ten sposób graf obrazuje niestrukturalne konstrukcje użyte w programie (dla programu w pełni strukturalnego, niezależnie od jego długości i liczby użytych instrukcji, złożoność zawsze wynosi 1). Złożoność esencjonalna bywa używana jako wskaźnik ostrzegawczy potencjalnych problemów z jakością kodu.

Złożoność danych (*data complexity*) może być rozpatrywana w dwojaki sposób. Po pierwsze, jako odnosząca się do liczby atrybutów związanych z poszczególnymi danymi. Tak rozumiana złożoność danych jest kluczowym czynnikiem wpływającym na poziom jakości danych. Niestety, nie opracowano dotychczas miary oceniającej złożoność danych, zatem pozostaje wyłącznie subiektywne szacowanie złożoności. Po drugie, złożoność danych można rozumieć jako sposób używania danych wewnątrz programu. W przeszłości zaproponowano kilka miar złożoności użycia danych, takich jak rozpiętość (*span*) [Elshoff 1976] czy też średnia liczba odwołań do zmiennych w instrukcji i średnia liczba tzw. aktywnych zmiennych w instrukcji [Dunsmore, Gannon 1979].

Złożoność powiązań międzymodułowych (*fan complexity*) odnosi się do liczby danych, które moduł programowy pobiera z innych modułów (*fan-in*), oraz liczby danych, które moduł przekazuje do innych modułów (*fan-out*) [Henry, Kafura 1981]. Jakość modułów o dużej liczbie *fan-out* ma znaczenie krytyczne, gdyż przekazują one wiele danych do dużej liczby innych modułów. Ale moduły o dużej liczbie *fan-in* też są ważne, gdyż są trudne do uruchomienia, albowiem są zależne od dużej liczby innych modułów zewnętrznych. Rozważając złożoność wynikającą z powiązań międzymodułowych, oprócz liczby przesyłanych danych należy uwzględnić również moc więzi międzymodułowych[2].

Złożoność stylu (*style complexity*) opiera się na ocenie łatwości lub trudności zrozumienia działania programu w zależności od tego, jaka jest jego czytelność. Czytelność programu nie ma żadnego znaczenia dla komputera – każdy poprawny składniowo program źródłowy po skompilowaniu jest jednakowo dobry dla komputera. Tymczasem obecność odpowiednio użytych pustych linii, komentarzy, wcięć tekstu, pogrupowania linii programu, właściwie dobranych identyfikatorów, zagnieżdżenia nawiasów w wyrażeniach itp. wpływa na czytelność programu – jego prostotę, jasność i przejrzystość. Złożoność stylu jest ważna dla ludzi, którzy będą poprawiali, modyfikowali i konserwowali program. A że czynności tego rodzaju pochłaniają znaczny odsetek budżetu każdego przedsięwzięcia informatycznego, to ważne jest, by można je było zrobić w sposób w miarę łatwy. Styl często traktowany jest jako pewna niemierzalna cecha programu, w związku z tym nie istnieją światowe standardy stylu. Istnieją jednak pewne powszechnie

[2] Problematykę mocy powiązań międzymodułowych poruszona została np. w [Myers 1980].

aprobowane zalecenia, cytowane przez wielu autorów[3]. Również wiele firm informatycznych wymaga od swych pracowników stosowania się do firmowych wytycznych stylu.

3. Podsumowanie

Jak wynika z przeprowadzonego wyżej przeglądu, pojęcie złożoności ma bardzo wiele znaczeń i przez to jego interpretacja jest trudna. Podkreślić przy tym należy, że w przypadku niektórych rodzajów złożoności jest ono względne (subiektywne) i to, co jeden obserwator uzna za złożone, inny może ocenić jako proste (tak jest np. w przypadku złożoności psychologicznej, problemu, organizacyjnej, procesu, funkcjonalnej). Z kolei stosując inne definicje złożoności, da się, przy zastosowaniu odpowiednio dobranych miar, obiektywnie ocenić złożoność (np. złożoność kodu, „teorii oprogramowania", cyklomatyczną, esencjonalną, danych). Oceniając złożoność we wczesnych fazach cyklu życia oprogramowania, najczęściej trzeba posłużyć się definicjami o dużej dozie subiektywizmu, a kiedy gotowy jest już kod programu – można ocenić złożoność w sposób bardziej obiektywny.

Niezależnie od tego, jak zostanie zdefiniowana, złożoność na zawsze pozostanie problemem, na który będą natrafiać twórcy oprogramowania, należy zatem nauczyć się radzić sobie z tym, co nieuniknione. Nie jest łatwo zaproponować jakieś skuteczne i uniwersalne środki zaradcze, ale można próbować podać pewne wskazówki. Powszechnie stosowana jest tu metoda rozkładania problemu lub systemu na mniejsze części, a następnie zrozumienia działania tych części i związków między nimi. Z tym mechanizmem abstrakcji, pozwalającym na budowę abstrakcyjnych obiektów i operowanie nimi bez wnikania w ich wewnętrzną budowę, bezpośrednio związany jest mechanizm ukrywania informacji, polegający na udostępnianiu zainteresowanemu tylko tych informacji, które mogą być dla niego istotne. Pozwala to programiście na skoncentrowanie się wyłącznie na najważniejszych w danym momencie sprawach, bez wnikania w szczegóły w danej chwili dla niego nieistotne. Wiąże się z tym dostępność narzędzi pozwalających ludziom na odszukanie w ogromie informacji o systemie tych danych, które są im w danym momencie rzeczywiście potrzebne. Trzecim aspektem jest ciągła restrukturyzacja produktu w taki sposób, by zredukować jego nadmierną złożoność.

Literatura

Błażewicz J., *Złożoność obliczeniowa problemów kombinatorycznych*, WNT, Warszawa 1988.
Brooks F.P. Jr., *Mityczny osobomiesiąc. Eseje o inżynierii oprogramowania*, WNT, Warszawa 2000.
Cardoso J., *Evaluating the Process Control-Flow Complexity Measure*, IEEE International Conference on Web Services (ICWS'05), 2005, s. 803-804.

[3] Obszerne omawia je Kobyliński [1991, s. 64-78].

Chall J.S., Dale E., *Readability revisited: The New Dale-Chall Readability Formula*, Brookline Books, Cambridge (MA) 1995.

Czarnacka-Chrobot B., *Metoda punktów funkcyjnych – bieżące standardy*, [w:] *Efektywność zastosowań systemów informatycznych*, t. 1, red. J.K. Grabara, J.S. Nowak, WNT, Warszawa – Szczyrk 2001, s. 57–83.

Dunsmore H.E., Gannon J.D., *Data Referencing: an Empirical Investigation*, „IEEE Computer Magazine" 1979, vol. 12, nr 12, s. 50-59.

Encyclopedia of Astrobiology, Astronomy and Spaceflight, http://www.daviddarling.info/encyclopedia/ETEmain.html, 2006-03-10.

Elshoff J.L., *An Analysis of Some Commercial PL/1 Programs*, „IEEE Transactions on Software Engineering" 1976, vol. SE-2, nr 2, s. 113-120.

Flank S., *Sentences vs. Phrases: Syntactic Complexity in Multimedia Information Retrieval*, [w:] *Syntactic and Semantic Complexity in Natural Language Processing Systems*, red. A. Bagga, J. Pustojevsky, W. Zadrozny, Seattle 2000.

Flesch R., *How to Write Plain English*, Harpercollins 1979.

Fry E., *Elementary Reading Instructions*, McGraw-Hill 1977.

Gunning R., *The Technique of Clear Writing*, McGraw-Hill 1952.

Halstead M.H., *Elements of Software Science*, North Holland, New York 1977.

Henry S.M., Kafura D.G., *Software Structure Metrics Based on Information Flow*, „IEEE Transactions on Software Engineering" 1981 nr 7(5), s. 510-518.

Kobyliński A., *Zagadnienia złożoności programów komputerowych*, rozprawa doktorska. Szkoła Główna Handlowa, Warszawa 1991.

Kobyliński A., *Problematyka złożoności w budowie systemów oprogramowania*, „Roczniki Kolegium Analiz Ekonomicznych" 2006, SGH (w druku).

Managing Organizational Complexity: Philosophy, Theory and Application, red. K.A. Richardson, Information Age Publishing 2005.

McCabe T.J., *A Complexity Measure*, „IEEE Transactions on Software Engineering" 1976, vol. SE-2, nr 4, s. 308-320.

Myers G.J., *Projektowanie niezawodnego oprogramowania*, WNT, Warszawa 1980.

Ndayambaje P., *Zagadnienia złożoności psychologicznej programów komputerowych*, rozprawa doktorska, Szkoła Główna Planowania i Statystyki, Warszawa 1990.

Neel D.L., Orrison M.E., *The Linear Complexity of a Graph*, „The Electronic Journal of Combinatorics", 2006, vol. 13, nr 1, http://cs.anu.edu.au/publications/eljc/Volume_13/PDF/v13i1r9.pdf, 2006-03-21.

Pollard S., Biermann A.W., *A Measure of Semantic Complexity for Natural Language Systems*, [w:] *Syntactic and Semantic Complexity in Natural Language Processing Systems*, red. A. Bagga, J. Pustojevsky, W. Zadrozny, Seattle 2000.

Shen V.Y., Conte S.D., Dunsmore H.E., *Software Science Revisited: a Critical Analysis of the Theory and Its Empirical Support*, „IEEE Transactions on Software Engineering" 1983, vol. SE-9, nr 2, s. 155-165.

Słownik języka polskiego, PWN, Warszawa 1989.

Watson A.H., McCabe T.J., *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*, NIST Special Publication 500-235, 1996, http://xsun.sdct.itl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/chaptera.htm, 2006-03-22.

Waxman M., *Definition of Problem Complexity*, 8.04.1996, http://bruce.edmonds.name/~bruce/waxman.txt, 2006-03-10.

SOFTWARE COMPLEXITY – INTERPRETATION OF THE NOTION

Summary

There are four aspects of software production that are inherently difficult: complexity, conformity, changeability and invisibility. In the paper the concept of "complexity" is thoroughly analyzed. The paper provides a short overview of different approaches to software complexity: psychological, code, semantic, syntactic, computational, algorithmic, problem, organizational, process, entropic, software science, function points, graph, cyclomatic, essential, data, fan and style complexity.

Dr inż. Andrzej Kobyliński jest kierownikiem Zakładu Technologii Oprogramowania Katedry Informatyki Gospodarczej Szkoły Głównej Handlowej w Warszawie, e-mail: Andrzej.Kobylinski@sgh.waw.pl.

Mgr Tomasz Chodola jest asystentem w Katedrze Informatyki Gospodarczej Szkoły Głównej Handlowej w Warszawie, e-mail: Tomasz.Chodola@sgh.waw.pl.