Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Škabar

Avtomatizacija ocene kompleksnosti izvorne programske kode

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

prof. dr. Miha Mraz
MENTOR

Ljubljana, 2016

Tematika naloge:

Kandidat naj v diplomskem delu opravi pregled metrik za oceno kompleksnosti izvorne programske kode. Pri tem naj razišče kakšna programska orodja so na voljo za avtomatizacijo postopka ocenjevanja in z izbranimi orodji opravi nekaj analiz na vzorčnih programih.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani izjavljam, da sem avtor dela, da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali drugem visokošolskem zavodu, razen v primerih kjer so navedeni viri.
S svojim podpisom zagotavljam, da:

- sem delo izdelal samostojno pod mentorstvom prof. dr. Mihe Mraza,
- so elektronska oblika dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko in
- soglašam z javno objavo elektronske oblike dela v zbirki "Dela FRI".

— Matjaž Škabar, Ljubljana, marec 2016.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Škabar

Avtomatizacija ocene kompleksnosti izvorne programske kode

POVZETEK

Cilj pričujočega diplomskega dela je preučitev metrik in orodij za analizo kompleksnosti izvorne programske kode ter primerjava različnih orodij med seboj. V uvodnem delu spoznamo problematiko na področju razvoja programske opreme in s kakšnimi težavami se podjetje ter programer lahko srečajo. V naslednjem poglavju spoznamo metrike s katerimi lahko zagotovimo, da tekom razvoja programske opreme posamezni sklopi programa ne postanejo prekompleksni za nadaljno nadgrajevanje ali uvajanje novih programerjev v projekt. Temu sledi seznam orodij od enostavnih, brezplačnih do zelo kompleksnih, plačljivih orodij. Pri vsakem od orodij spoznamo, katere metrike zna izračunati, v zadnjem poglavju pa je predstavljena izvedena avtomatizirana analiza vzorčnih primerov izvornih programskih kod. Poročila različnih orodij so med seboj primerjana, kar koristi pri izbiri pravega orodja za avtomatizirano analizo kompleksnosti izvorne programske kode pri razvoju programske opreme.

Ključne besede: avtomatizacija, programska metrika, programska analiza, McCabe, Halstead, LOC

University of Ljubljana
Faculty of Computer and Information Science

Matjaž Škabar

Automation of estimating source code complexity

ABSTRACT

The goal of this thesis is to examine metrics and tools for analyzing the complexity of program source code and to compare different tools with each other. In the opening part, the issues in the field of software development and problems the company and the programmer might encounter are presented. In the next chapter, we learn about the metrics with which we can ensure that during software development individual parts of the program do not become too complex for further upgrading or for introducing new programmers to the project. This is followed by a list of tools, ranging from simple free to very complex paid tools. It is examined which tool calculates which metric and in the last chapter an automated analysis of samples of source code is conducted. The comparison of the reports about these tools will be helpful when choosing the right tools for automated analysis of the complexity of the software source code in software development in the future.

Key words: automation, software metrics, software analysis, McCabe, Halstead, LOC

ZAHVALA

Najprej se zahvaljujem svoji družini, ki me že od majhnega spodbuja in podpira pri izobraževanju in mi je omogočila študij na fakulteti.

Iskreno se zahvaljujem mentorju prof.dr.Mihu Mrazu za potrpežljivost, vloženo delo pri pregledovanju naloge in odlično strokovno svetovanje.

Hvala tudi sodelavcem in prijateljem za motivacijo in razumevanje.

Rad bi se zahvalil tudi vsem podjetjem, ki so mi omogočila uporabo svojih orodij s posredovanjem evaluacijskih kod za svoje programe.

Za konec se zahvaljujem še moji Mariji za vso spodbudo, podporo in potrpežljivost.

— Matjaž Škabar, Ljubljana, marec 2016.

KAZALO

1 Uvod

Današnji programski jeziki omogočajo razvoj programov, katerih kompleksnost lahko hitro naraste in postane neobvladljiva. Koda, ki je preveč kompleksna, bo zagotovo predstavljala težave pri nadgrajevanju ter odpravljanju napak, saj obstaja velika verjetnost, da programer, ki bo v prihodnosti urejal to kodo, te ne bo razumel. Težave predstavljajo tudi cikli odvisnosti, zaradi katerih posameznega sklopa ne moremo testirati, nadgrajevati in spreminjati brez vključitve sklopa, od katerega je opazovani sklop odvisen. Do teh težav zlahka pride v primeru, ko se za izdelavo določenih sklopov programske kode najame zunanje izvajalce, ki včasih vrnejo slabo dokumentirano kodo. Kdor bo takšno kodo nadgrajeval, se bo zagotovo srečal s kakšno od težav, ki jih zaznavajo v tem delu opisane metrike in orodja. Problem predstavljajo tudi enojne točke izpada SPOF (angl. *single point of failure*). To pomeni, da so za projekt ključni določeni programerji, katerih odhod lahko kritično ogrozi nadaljni razvoj ali celo obstoj projekta.

Vse to zelo podraži razvoj programske opreme v prihodnosti. Programerji in podjetja, ki želijo razvijati kvalitetno izvorno kodo, zato stremijo k temu, da uporaba avtomatiziranih orodij za analiziranje izvorne kode postane del vsakdana, saj ta orodja bistveno

pripomorejo k kvalitetnejši kodi. V prejšnem odstavku omenjenim točkam odpovedi se lahko izognemo na primer z rabo orodij za analiziranje izvorne programske kode, saj na ta način izvemo, ali je koda dokumentirana in kakšna je McCabova kompleksnost metod. Na podlagi vzdrževalskega indeksa se lahko podjetje odloči, ali bo določen sklop razvilo znova, ali pa je možno nadgraditi staro kodo, itd. V primeru, da podjetje izvaja enotske teste, lahko na podlagi rezultatov avtomatiziranih analiz McCabove ciklomatične kompleksnosti programer izve, najmanj koliko enotskih testov mora narediti, na podlagi Halsteadove metrike pričakovanega števila napak v kodi pa izve, koliko napak naj bi našel v kodi, da zagotovi čimvečjo pravilnost izvorne kode.

Na voljo je mnogo programskih orodij, od integriranih v razvojna okolja do brezplačnih in plačljivih orodij. Izbira ustreznega orodja predstavlja težavo. Poleg nabora informacij, ki jih orodje nudi, so razlike tudi v kakovosti in strukturi analitičnih poročil. Nekatera poročila so zelo pregledna in informacij ni potrebno iskati, različne sklope pa lahko zlahka primerjamo med seboj. Druga pa so strukturirana tako, da je potrebno informacije iskati, neposredna primerjava med različnimi sklopi programa pa je nemogoča. Razlika med orodji je tudi v tem, ali orodje omogoča integracijo v razvojno okolje, ali pa je potrebno analize zagnati ločeno.

Za cilj diplomske naloge smo si zadali testiranje ter analizo rezultatov različnih orodij, da bi ugotovili, katero orodje je najprimernejše za avtomatizacijo ocene kompleksnosti izvorne programske kode. Pomembno je, da je programer seznanjen z različnimi metrikami, ki so na voljo za analiziranje izvorne programske kode, zato bomo najprej spoznali metrike, ki so danes v uporabi, da bi bilo v kasnejših poglavjih razumevanje funkcionalnosti orodij in poročil analiz lažje. V nadaljevanju so predstavljena orodja, ki s pomočjo teh metrik analizirajo izvorno programsko kodo. Spoznali bomo razlike med orodji, našteli orodja, ki omogočajo integracijo v razvojna okolja in pri vsakem orodju našteli metrike, ki jih to orodje zna izračunati. V zadnjem poglavju so predstavljeni rezultati vzorčnih analiz. Videli bomo tudi, ali različna orodja metrike interpretirajo na enak način, ali pa se lahko pri rezultatih analiz pojavijo razlike.

Dobljene informacije bodo pomagale pri izbiri orodja za analiziranje določenega projekta. Izvedeli bomo, katero orodje je dovolj dobro za določen projekt, katero analizira dovolj metrik in katero orodje rezultate analize poda v dobro strukturiranem poročilu.

2 Metrike za ocenjevanje kompleksnosti izvorne kode

V pričujočem poglavju predstavimo pregled *kvantitativnih* in *kvalitativnih* metrik za oceno kompleksnosti programske opreme. Obe vrsti metrik lahko razdelimo na statične in dinamične. Pri merjenju *statičnih* programa ne izvajamo, pri merjenju *dinamičnih* pa se program izvaja.

Z metrikami lahko testiramo posamezne metode, datoteke, celotne projekte ali pa le določen paket. Pomeni posameznih naštetih pojmov so sledeči:

- Paket zajema vse, kar je nižje od trenutnega pogleda. Aplikacija na primer zajema sklope (angl. *assembly*).
- Sklop zajema imenske prostore (angl. *namespace*). Imenski prostori vsebujejo set povezanih objektov. S pomočjo imenskih prostorov lahko naredimo več različnih tipov.
- Tip je lahko nov imenski prostor, razred, vmesnik, „struct“, enumerator ali delegat.
- Znotraj tipov se nahajajo metode in polja. Vsebina paketa je torej odvisna od tega, na kateri točki v drevesni strukturi izvorne kode se nahajamo.

2.1 Preštevalne metrike

Preštevalne metrike omogočajo vpogled v količino izvorne kode in s tem posredno v njeno kompleksnost.

2.1.1 Preštevanje vrstic izvorne kode

LOC (angl. *Lines of code*) metrika je statična metrika, kjer kode ne izvajamo. Predstavlja število vrstic izvorne kode [13]. Slednje posredno odraža kompleksnost programa. Moderna razvojna okolja lahko zaradi svoje kompleksnosti tovrstno metriko naredijo manj primerno za določanje kompleksnosti programov [16]. Štetje vrstic izvorne kode delimo na dve skupini:

- *Štetje fizičnih vrstic kode*: Metoda šteje fizične vrstice kode. Slednje imajo jasen začetek in konec in niso odvisne od jezika, zato takšno štetje uvrščamo med preprostejše metrike. To štetje ne upošteva sintaktičnih in ostalih razlik med jeziki, zato velja, da je metrika povsem neodvisna od programskega jezika. Na tej točki moramo biti pozorni predvsem na različne interpretacije štetja *LOC* med različnimi programskimi orodji, saj kljub velikemu prizadevanju IEEE (angl. *Institute of Electrical and Electronics Engineers*) razvijalci števnih orodij še niso prišli do konsenza o univerzalnem štetju fizičnih *LOC*. Poznamo naslednje načine štetja vrstic:
  - število fizičnih vrstic izvorne kode **LOC** (angl. *lines of code*): vsebuje tako izvorno kodo kot tudi komentarje [3],
  - število praznih vrstic izvorne kode **BLOC** (angl. *blank source lines of code*): navadno služijo preglednosti in berljivosti kode [3],
  - število komentarjev **CLOC** (angl. *commented source lines of code*): število vrstic s komentarji [3], kar nam da predstavo o dokumentiranosti kode; večje kot je število komentarjev, večja je verjetnost, da bo prihodnje popravljanje ali nadgrajevanje kode lažje; podjetje Testwell priporoča, da se delež komentarjev glede na število vseh vrstic izvorne kode giblje med 30 in 75 odstotki; v kolikor je datoteka komentirana manj kot 30 odstotkov se lahko sklepa, da je slabo ali le trivialno dokumentirana, medtem ko koda, ki je komentirana več kot 75 odstotno niti ni več izvorna koda temveč dokument [16],

- število besed v komentarjih **CWORDS** (angl. *commented words*): obseg komentarjev [3],
- število vrstic izvorne kode **SLOC** (angl. *source lines of code*): število vrstic izvorne kode brez praznih vrstic in komentarjev [3].

- *Štetje logičnih vrstic kode*:
  - število logičnih vrstic kode **SLOC-L** (angl. *logical source lines of code*): vsebuje vrstice z operacijami in operandi [3].

2.1.2 Ostale metrike preštevanja

Poleg štetja vrstic kode poznamo še metrike, ki omogočajo štetje naslednjih parametrov:

- tipov, polj, parametrov, spremenljivk,
- metod, funkcij,
- razredov, datotek in sklopov (angl. *assembly*),
- paketov, imenskih prostorov (angl. *namespace* je ključna beseda v programskem jeziku C#, ki določa obseg povezanih objektov, razredov; uporablja se za urejanje kode in ustvarjanje globalnih edinstvenih tipov [12]),
- programov, rutin, klinov,
- definicij, privzetih imen, edinstvenih označb tipov,
- navodil za premike,
- dohodnih odvisnosti *CA* (angl. *afferent coupling*): gre za število razredov drugega paketa B, ki so odvisni od razredov v paketu A; število predstavlja pomembnost razreda,
- odhodnih odvisnosti *CE* (angl. *efferent coupling*): gre za število razredov paketa A, ki so odvisni od razredov v drugem paketu B; število predstavlja odvisnost razreda od drugih razredov,
- število otrok razreda *NOC* (angl. *number of children*) je število podrazredov, ki ta razred dedujejo,
- globina dedovanja *DIT* (angl. *depth of inheritance tree*) za določen razred ali strukturo je število razredov, ki jih ta razred deduje.

2.2 Računske metrike

Računske metrike omogočajo vpogled v posamezne module, funkcije in podsisteme, na podlagi česar lahko sklepamo, kakšna je kakovost izvorne kode. To razvijalcem omogoča odkrivanje kritičnih predelov izvorne kode z možnostjo ukrepanja. Prevelika kompleksnost funkcij namreč pomeni neposredno nepreglednost kode in funkcij ter s tem bistveno oteženo nadgrajevanje in razumevanje izvorne kode v prihodnosti.

2.2.1 Halsteadova metrika

Halsteadova metrika je statična metrika, kjer kode ne izvajamo. Služi za oceno števila napak v programu [1], kar je tudi ciljno število napak, ki jih želimo odkriti v analizirani kodi. Za meritev uporabimo:

- operatorje (tradicionalne ukaze in ključne besede) in operande (spremenljivke: identifikatorje in konstante),
- LOC, $SLOC$ in $LLOC$,
- enote, datoteke, razrede,
- pakete, mape.

Halstead je za izračun metrik definiral naslednje spremenljivke:

- n_1: število različnih operatorjev,
- n_2: število različnih operandov,
- N_1: število vseh operacij,
- N_2: število vseh operandov.

S pomočjo navedenih spremenljivk lahko izračunamo sledeče metrike [14]:

- dolžino programa N

$$N = N_1 + N_2, \tag{2.1}$$

- slovar programa n

$$n = n_1 + n_2, \tag{2.2}$$

- prostornina programa V

$$V = N * log_2 n, \tag{2.3}$$

- kompleksnost programa D

$$D = (n_1/2) * (N_2/n_2), \tag{2.4}$$

- vložek v kodo E

$$E = D * V, \tag{2.5}$$

- čas za razumevanje/implementacijo v sekundah T

$$T = E/18, \tag{2.6}$$

- pričakovano število napak B

$$B = N^{2/3}/3000, \tag{2.7}$$

- stopnjo programa L

$$L = 1/D. \tag{2.8}$$

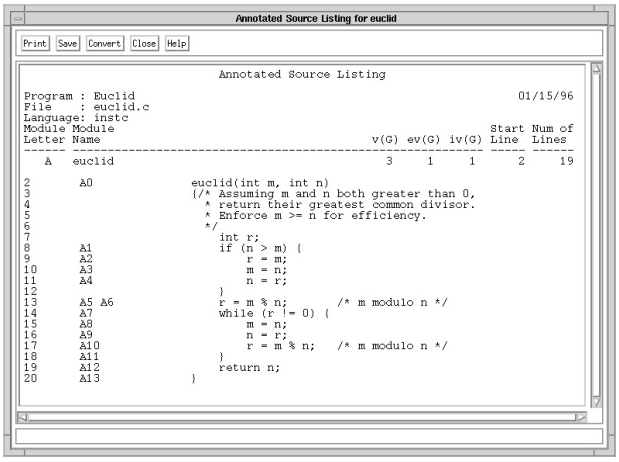
2.2.2 McCabova ciklomatična metrika je slednje:

McCabova ciklomatična metrika (angl. *McCabe Cyclomatic Complexity Metric*) je statična metrika, kjer kode ne izvajamo. Metrika izmeri število alternativnih poti v programu in s tem neposredno izraža kompleksnost vejitev v programu (Mc Cabe-ovo ciklomatično število [14]). Velja, da, večja kot je kompleksnost, težje razumljiva, nadgradljiva in popravljiva je koda. Obenem nam vrednost $v(G)$ poda minimalno število testov, ki jih moramo oblikovati in izvesti, da zagotovimo pokritost analize celotne kode v funkciji ali modulu.
Za izračun ciklomatične kompleksnosti moramo program predstaviti z grafom G, ki ima natanko eno vhodno in eno izhodno točko, nato pa uporabimo enačbo [10, 13, 14]

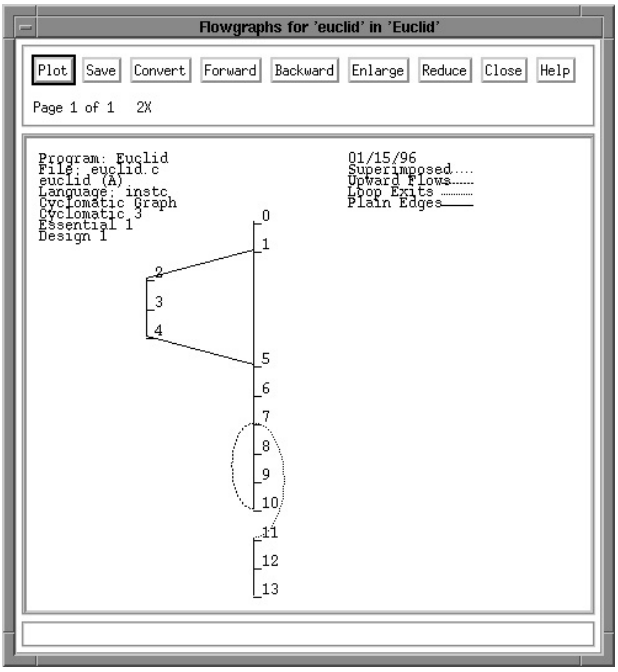
$$v(G) = E - N + 2, \tag{2.9}$$

kjer $v(G)$ predstavlja izračunano kompleksnost, E število povezav med vozlišči in N število vozlišč v grafu [10]. Ključni pomen pri izračunu vrednosti ciklomatične metrike imajo odločitve v funkciji (if, for, while, case, zanke, try/catch izjave) [9].

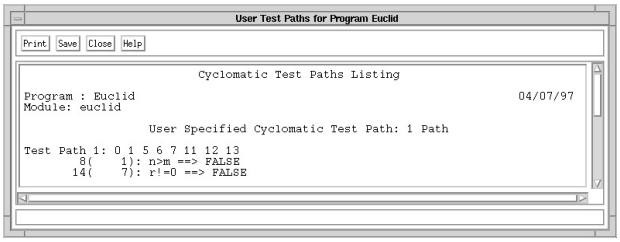
Primer izračuna McCabove ciklomatične metrike Evklidovega algoritma je prikazan na slikah 2.1, 2.2 in 2.3.



Slika 2.1 Implementacija Evklidovega algoritma v programskem jeziku C [10].



Slika 2.2 Diagram poteka Evklidovega algoritma prikazanega na sliki 2.1 v programskem jeziku C [10].



Slika 2.3 Primer poti, ki gre skozi vsa vozlišča (v primeru, da ne bi bilo mogoče potovati skozi vsa vozlišča, bi morali navesti več možnih poti) za program na sliki 2.1 [10].

S pomočjo prej omenjene enačbe ugotovimo, da je $v(G)$ za dani primer 3.

$$v(G) = 15 - 14 + 2 \tag{2.10}$$

Zaželjeno je, da je kompleksnost programa čim manjša. Po izvedbi opazovalnih testov je McCabe predlagal, da je zgornja meja kompleksnosti $v(G)$ največ 10, sicer je funkcija preveč kompleksna za razumevanje in nadaljni razvoj [10]. Andersson je kasneje za najvišjo sprejemljivo vrednost kompleksnosti $v(G)$ predlagal vrednost 20 [2], saj po njegovem mnenju prekoračitev te vrednosti znatno zviša verjetnost vnešenih napak [2, 7]. Danes tako velja, da naj bi bila kompleksnost funkcij največ med 10 in 20 [14]. Na podlagi teh meja lahko hitro najdemo funkcije, kjer je največja verjetnost pojavitve težave pri načrtovanju enotskih testov za dotično funkcijo ter pri razumevanju in nadaljnjem razvoju funkcije.

S pomočjo ciklomatične kompleksnosti lahko izračunamo tudi gostoto odločitev. To je ciklomatična kompleksnost na vrstico izvorne kode.

2.2.3 Vzdrževalski indeks

Vzdrževalski indeks MI (angl. *maintainability index*) je statična metrika, kjer pridobljeno oceno pridobimo eksperimentalno na podlagi Halsteadove metrike truda ali obsega, McCabove ciklomatične metrike, števila vrstic izvorne kode (LOC) in števila komentarjev [2, 14]. Namenjen je zmanjšanju entropije ali degradiranju integritete kode. Za izračun indeksa vzdrževalnosti MI moramo sešteti rezultata dveh enačb in sicer enačbe, ki komentarje upošteva, in enačbe, ki komentarjev ne upošteva [2, 14]:

- enačba, ki ne upošteva komentarjev:

$$MI_1 = 171 - 5.2ln(V) - 0.23v(G) - 16.2ln(LOC) \tag{2.11}$$

- pomen parametrov pri tej enačbi je naslednji:
  - *ln(V)* ... logaritem Halsteadove metrike,
  - *v(G)* ... McCabe-ova metrika,
  - *ln(LOC)* ... logaritem števila vrstic kode,

- enačba, ki upošteva komentarje:

$$MI_2 = 171 - 5.2ln(V) - 0.23v(G) - 16.2ln(LOC) + 50sin\sqrt{2.46perCM} \quad (2.12)$$

- enačbo (2.12) lahko poenostavimo na naslednji način:

$$MI_2 = MI_1 + 50sin\sqrt{2.46perCM} \quad (2.13)$$

  - pomen parametrov pri tej enačbi je naslednji:
    - *ln(V)* ... logaritem Halsteadove metrike,
    - *v(G)* ... McCabe-ova metrika,
    - *ln(LOC)* ... logaritem števila vrstic kode,
    - *perCM* ... odstotek komentarjev na funkcijo,
    - MI_2 je uporaben samo v primeru dobro komentirane kode; pri tem pomembno vlogo igra človek, ki oceni kvaliteto komentarjev;

- da dobimo vzdrževalni indeks, moramo dobljena rezultata enačb (2.11) in (2.12) sešteti:

$$MI = MI_1 + MI_2 \quad (2.14)$$

Dobljene vzdrževalske indekse lahko razporedimo v naslednje kategorije [7]:

- $MI > 85$ ... z vzdrževanjem kode ne bo večjih težav in se jo lahko nadgrajuje,
- $85 =< MI =< 65$ ... z vzdrževanjem kode lahko pričakujemo nekaj težav in se jo lahko nadgrajuje,
- $MI < 65$ ... pri vzdrževanju kode se bodo zagotovo pojavile težave in je priporočljivo, da se jo razvije znova.

Na podlagi dobljenih rezultatov vzdrževalnega indeksa MI se podjetja lahko odločijo, ali se del programa zaradi težavnosti nadaljnega razvoja razvije znova, ali pa se lahko obstoječe funkcije razvija dalje.

2.2.4 Enotski testi

Enotski test je ogrodje za avtomatizacijo testiranja programske kode (angl. *McCabe Test Harness*). To ogrodje je programski paket, s katerim programer testira, ali programska koda vrne željeni rezultat in ali se programska koda izvaja na pričakovan način. Enotski test (angl. *Unit test harness*) mora uporabniku omogočati sledeče funkcije [6] :

- skupni jezik za izražanje testnih primerov (testni primer opiše točna pričakovanja, kako se bo koda obnašala med testiranjem; s tem ustvarimo predpogoje, na podlagi katerih lahko kasneje preverimo izpolnitev preostalih pogojev),
- skupni jezik za izražanje pričakovanih rezultatov,
- dostop do funkcionalnosti programskega jezika produkcijske kode (kode, ki je ali bo uporabljena v končnem produktu),
- zbiranje vseh enotskih testov projekta, sistema in podsistema,
- mehanizem za poganjanje celotnih ali delnih serij enotskih testov,
- razumljivo in jedrnato poročilo o uspehu ali neuspehu testa,
- podrobno poročilo v primeru neuspelega testa.

2.2.5 Preostale računske metode

Iz predhodno obravnavanih metrik lahko posredno izračunamo naslednje vrednosti:

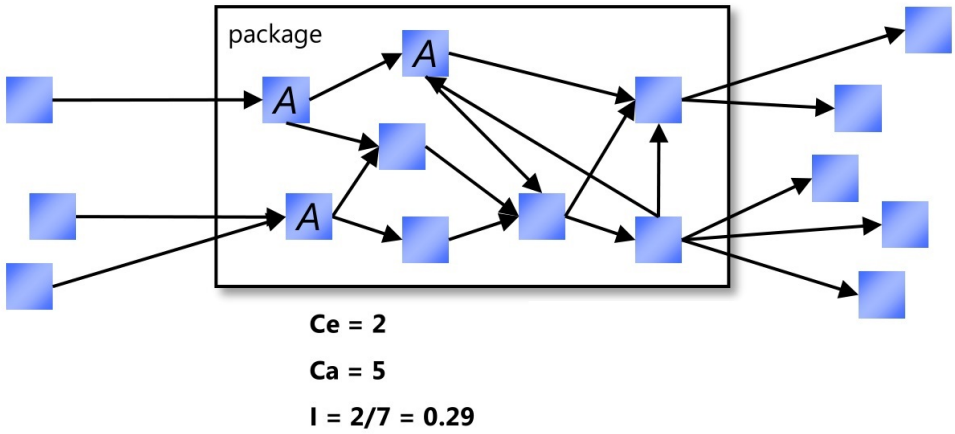
- gostoto gnezdenih zank,
- največjo gostoto gnezdenih pogojnih stavkov,
- odstotek komentarjev glede na celotno kodo,
- odstotek pokritosti testa,
- nestabilnost I, ki predstavlja razmerje med odhodno in dohodno odvisnostjo razreda; izračunamo jo z enačbo

$$I = CE/(CE + CA), \tag{2.15}$$

kjer CA pomeni število razredov, ki so odvisni od zunanjih paketov, CE pa število zunanjih paketov, ki so odvisni od razredov analiziranega paketa (razlago CA in CE najdemo v poglavju 2.1.2).

- abstraktnost A predstavlja razmerje med abstraktnimi tipi in vsemi tipi znotraj paketa.

Primer izračuna nestabilnosti lahko vidimo na sliki 2.4. Na sliki vidimo, da sta dva razreda v našem paketu odvisna od zunanjih paketov medtem ko je 5 zunanjih paketov odvisnih od analiziranega paketa.



Slika 2.4 Prikaz grafa paketov in njihove medsebojne odvisnosti ter izračuna nestabilnosti [4].

Izračunamo lahko tudi naslednje vrednosti:

- relacijsko kohezijo H; to je povprečno število notranjih povezav med tipi; izračunamo jo z enačbo

$$H = (R + 1)/N, \tag{2.16}$$

  kjer R predstavlja število število povezav med tipi, N pa predstavlja število tipov v paketu;

- abstraktnost A; razmerje med številom internih abstraktnih razredov in vmesnikov ter številom vseh razredov v analiziranem paketu; razpon razmerja je med 0 in 1, kjer 0 pomeni trden, 1 pa povsem abstrakten sklop;

- rang $R(A)$; izračunamo ga z enačbo

$$R(A) = (1 - d) + d \sum_{i=1}^{n} R(Ti)/CA(Ti), \tag{2.17}$$

- pomanjkanje kohezije metod $LOCM$; kohezija metod se drži načela edinstvene odgovornosti, ki pravi, da ima razred eno samo odgovornost in da ima lahko največ en razlog, da se spremeni; izračunamo jo z enačbo

$$LOCM = 1 - (\Sigma_{f \epsilon F} |M_f|)/(|M| x |F|), \tag{2.18}$$

  pomen parametrov pri tej enačbi je naslednji:

  - M ... statične metode in inštance metod v razredu,
  - F ... inštance polj v razredu,
  - M_f ... metode, ki dostopajo do polj f.

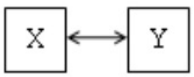
2.3 Preostale metode

Poleg predhodno navedenih metrik je pri razvoju programske opreme pomembno, da smo pazljivi tudi na cikle odvisnosti, saj imajo ti velik vpliv tako na razvoj izvorne kode, kot tudi na testiranje.

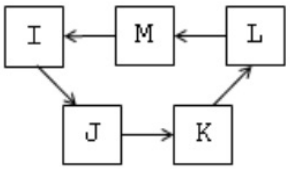
2.3.1 Cikli odvisnosti

Cikli odvisnosti navadno povzročijo zelo nepregledno kodo. Cikel odvisnosti na primer pomeni, da je komponenta A odvisna od komponente B in je v nadaljnem komponenta B odvisna od komponente C in je v nadaljnem komponenta C odvisna od komponente A. Takšen cikel povzroči, da komponente A ne moremo neodvisno testirati in razvijati, saj skupek enot A, B in C tvori nevidno enoto. To povzroči bistveno višje stroške razumevanja in razvoja programske kode [15].

Na slikah 2.5 in 2.6 sta prikazana primera ciklične odvisnosti.



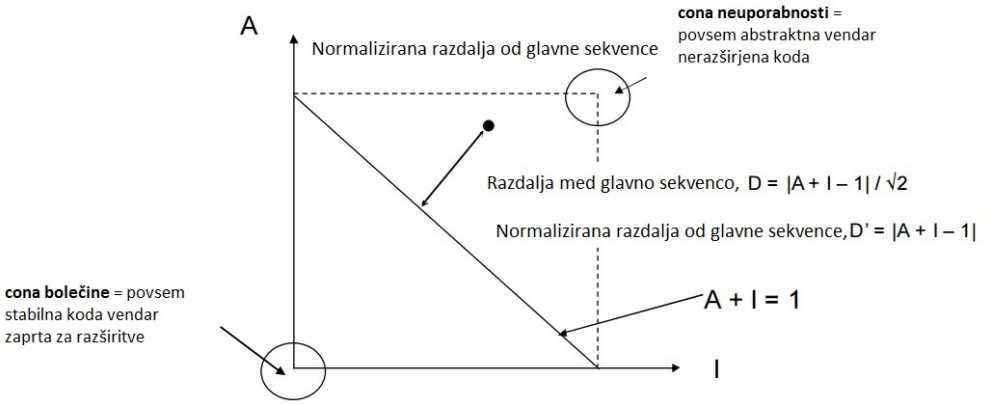
Slika 2.5 Prikaz ciklične odvisnosti, kjer je X odvisen od Y in Y odvisen od X [11].



Slika 2.6 Prikaz ciklične odvisnosti, kjer je I odvisen od J, J odvisen od K, K odvisen od L, L odvisen od M in M odvisen od I [11].

2.3.2 Normalizirana oddaljenost od glavnega zaporedja

Normalizirana oddaljenost od glavnega zaporedja D (angl. *distance from main sequence*) je oddaljenost paketa od idealne linije v grafu $A + I = 1$. Ta linija predstavlja optimalno ravnovesje med abstraktnostjo ter stabilnostjo sklopa. Vrednost 0 pomeni, da je sklop v idealnem ravnovesju, medtem ko so vrednosti med $0.7 < D < 1$ problematične [4, 5]. Graf normalizirane oddaljenosti od glavnega zaporedja je predstavljen na sliki 2.7. Na sliki I predstavlja nestabilnost, A pa predstavlja abstraktnost. Razlago obeh najdemo v poglavju 2.2.5.

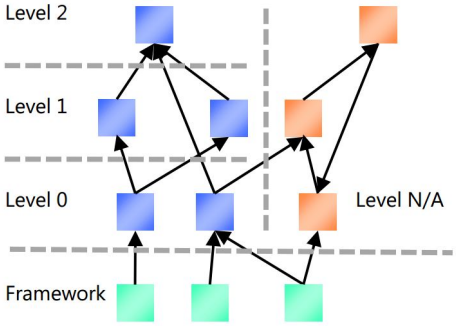


Slika 2.7 Prikaz grafa normalizirane odaljenosti od glavnega zaporedja [5].

2.3.3 Stopnja paketa

Stopnja paketa (angl. *level of package*) je odvisna od odvisnosti paketa (angl. *package)* od drugih paketov. V primeru, da paket ni odvisen od drugih paketov, ali pa je odvisen samo od ogrodja (angl. *framework*), je ta paket stopnje 0. V primeru, da je odvisen od paketov stopnje N, torej paketov, ki so odvisni od N drugih paketov, je stopnja tega

paketa $N+1$. V primeru, da je paket del cikla odvisnosti je njegova stopnja N/A (angl. *non applicable*). Če je paket odvisen od paketa, ki je stopnje N/A, je tudi stopnja tega paketa N/A [4].
Na sliki 2.8 je prikazana shema stopenj paketa.



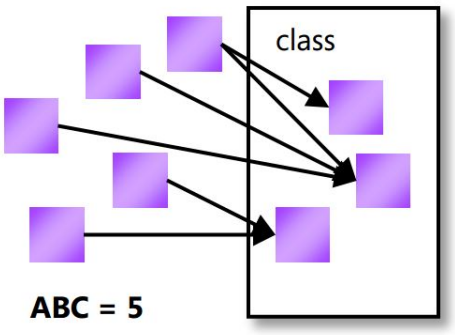
Slika 2.8 Prikaz stopnje paketa [4].

2.3.4 Velikost primerka

Velikost primerka (angl. *size of instance*) polja je enaka velikosti primerka razreda ali ogrodja izražena v bajtih. Velikost statičnega primerka je 0. Velikost primerka razreda ali strukture je seštevek velikosti primerkov polj k čemur prištejemo velikost primerka osnovnega razreda [4].

2.3.5 Asociacije med razredi

Vrednost asociacije med razredi ABC (angl. *association between classes*) za določen razred ali strukturo je število članov drugih razredov ali ogrodij, ki jih ta razred uporablja v svojih metodah [4]. Na sliki 2.9 je prikazana vrednost asociacije med razredi.



Slika 2.9 Prikaz vrednosti asociacije med razredi [4].

3 Pregled orodij za avtomatizacijo ocene kompleksnosti kode

Za avtomatizacijo ocene kompleksnosti programske kode obstaja mnogo programskih orodij. Za namene testiranja in raziskovanja sem izbral nekaj enostavnejših brezplačnih in nekaj kompleksnejših plačljivih orodij. Brezplačna orodja so kljub svoji enostavnosti uporabna, saj uporabniku vrnejo osnovne informacije o določenih sklopih izvorne kode ali celotnega projekta, plačljiva orodja pa svojo vrednost uporabniku vrnejo s podajanjem informacij, ki pomagajo pri iskanju in reševanju tako trenutnih napak, kot tudi morebitnih težav pri nadaljnem razvoju programske kode.

3.1 LocMetrics

Prednost orodja LocMetrics je predvsem njegova preprostost, poleg tega pa je na voljo brezplačno. Uporabno je za analiziranje posameznih ali več datotek programov pisanih v jezikih C#, C++, Java in SQL. Rezultati analize so seštevki vseh metrik za vse izbrane datoteke. Orodje žal ne omogoča projektne analize. V tabeli 3.1 so prikazane metrike, katerih izračun orodje omogoča [8]:

METRIKE, KI JIH ORODJE OMOGOČA	
Vrste metrik	Metrike
Preštevanje vrstic izvorne kode	Štetje vrstic z izvorno kodo *LOC* Štetje praznih vrstic izvorne kod *SLOC-B* Štetje logičnih vrstic izvorne kode *SLOC-L* Štetje vrstic s samo izvorno kod *SLOC* Štetje vrstic kode s komentarji *CLOC* Štetje besed v komentarjih *CWORDS*
Ostale metrike preštevanja	Štetje datotek z izvorno kodo Štetje map z izvorno kodo Štetje vrstic kode s komentarji v glavi datoteke Štetje besed v komentarjih v glavi datoteke
Računske metrike	McCabeova ciklomatična metrika

Tabela 3.1 Metrike, ki jih omogoča orodje LocMetrics.

Program LocMetrics vse izračune zabeleži na nivoju testirane datoteke, kasneje pa sešteje vrednosti predhodno navedenih spremenljivk za celoten projekt in izriše pregleden histogram.

3.2 Semantic Designs CSharp Software Metrics

Orodje Semantic Designs CSharp Software Metrics je kljub svoji preprostosti zelo močno orodje. Tako kot orodje LocMetrics tudi Semantic Designs CSharp Software Metrics ne omogoča analize projektov temveč le izbranih datotek. Za razliko od orodja LocMetrics Semantic Designs CSharp Software Metrics poročilo poda strukturirano, tako da vidimo

podatke tako za celoten sklop testiranih datotek izvorne kode, kot tudi za posamezno datoteko in metodo.

Orodje lahko uporabimo za analizo programov pisanih v jezikih C#, COBOL, Java, Logix5000 (IEVER 2.6) in VBScript (Beta). Funkcije, ki jih orodje ponuja, so različne glede na različne programske jezike [17]. V tabeli 3.2 so prikazane vse metrike, ki jih orodje omogoča.

METRIKE, KI JIH ORODJE OMOGOČA		
Vrste metrik	Metrike	
Preštevanje vrstic izvorne kode	Štetje vrstic z izvorno kodo *LOC* Štetje praznih vrstic izvorne kode *SLOC-B* Štetje logičnih vrstic izvorne kode *SLOC-L* Štetje vrstic s samo izvorno kodo *SLOC* Štetje vrstic kode s komentarji *CLOC* Štetje besed v komentarjih *CWORDS*	
Ostale metrike preštevanja	Štetje datotek z izvorno kodo Štetje tipov Štetje metod	
Računske metrike	McCabeova ciklomatična metrika	Seštevek vrednosti vseh funkcij Povprečje vrednosti vseh funkcij Mediana vrednosti vseh funkcij
	Halsteadova metrika	Število edinstvenih operatorjev Število edinstvenih operandov Število pojavitev operatorjev Število pojavitev operandov Slovar programa n Dolžina programa N Volumen programa V Kompleksnost programa D Vložek v kodo E Pričakovano število napak B
	Vzdrževalski indeks *MI*	

Tabela 3.2 Metrike, ki jih omogoča orodje Semantic Designs CSharp Software Metrics.

Orodje omogoča različne funkcionalnosti za različne programske jezike. Naštejmo glavne funkcionalnosti:

- analiza operacij, ki se izvajajo nad specificiranimi datotekami,

- meritve izračunane na nivoju modula/funkcije, divizije, razdelka, odstavka, datoteke, kontrolerja ali rutine,

- identifikacija in meritve najslabših primerov gnezdenj z osredotočenostjo na posamezno datoteko ali vrstico izvorne kode,

- identifikacija in meritve najslabših primerov gnezdenj, osredotočena na posamezno datoteko, vrstico izvorne kode, razred ali paket,

- povzetek na nivoju razreda ali datoteke,

- poročilo v obliki teksta, XML datoteke.

3.3 Testwell CMT++, CMTJava

Orodji Testwell CMT++ in CMTJava, ki ju distribuira podjetje Verifysoft, se ponašata z dobrim vmesnikom programa, ki uporabniku omogoča pregledno izbiranje vhodnih datotek ter način izpisa poročila. Slednje v tekstovni obliki, ki jo izvozi v datoteko ali na standardni izhod, v HTML obliki ali v Excel oziroma XML datoteki. HTML oblika je dobro strukturirana, zaradi česar je pregledovanje poročila enostavno ter pregledno kljub njegovi potencialni veliki obsežnosti. Orodje je zmožno analizirati programe, ki obsegajo preko milijona vrstic izvorne kode.
Namenjeno je iskanju sumljivo kompleksnih delov izvorne kode. Na podlagi McCabeove ciklomatične vrednosti in Halsteadove ocene vnešenih napak v kodi izvemo koliko enotskih testov moramo izdelati in koliko hipotetičnih napak moramo najti v kodi. Izvemo tudi, katere enote programa predstavljajo večje tveganje. Ta ocena verjetnosti je neposredno povezana s kompleksnostjo programske kode. V tabeli 3.3 so prikazane metrike, ki jih orodje omogoča.

METRIKE, KI JIH OMOGOČA ORODJE CMT++		
Vrste metrik	Metrike	
Preštevanje vrstic izvorne kode	Štetje vrstic z izvorno kodo *LOC*	
	Štetje praznih vrstic izvorne kode *SLOC-B*	
	Štetje vrstic kode s komentarji *CLOC*	
Ostale metrike preštevanja	Štetje vrstic izvorne kode z deklaracijami, definicijami, direktivami	
Računske metrike	McCabeova ciklomatična metrika	Izračunana na nivoju razreda in metode
		Izračunana na nivoju celotne datoteke
	Halsteadova metrika	Število edinstvenih operandov
		Število edinstvenih operatorjev
		Število pojavitev operandov
		Število pojavitev operatorjev
		Slovar programa n
		Dolžina programa N
		Volumen programa V
		Kompleksnost programa D
		Vložek v kodo E
		Pričakovano število napak B
		Stopnja programa L
		Čas za razumevanje/implementacijo v sekundah T
	Vzdrževalski indeks *MI*	

Tabela 3.3 Metrike, ki jih omogoča orodje CMT++.

3.4 NDepend

NDepend prav gotovo sodi med kompleksnejša orodja za izdelavo analize programske kode. Omogoča nam celovito oceno kompleksnosti izvorne kode in določanje vseh v tem delu opisanih metrik, poleg tega pa omogoča tudi integracijo z orodjem Visual Studio, zaradi česar lahko analiziramo tudi razvoj projektov in rešitev. S tem dobimo sproten vpogled v stanje, strukturo in kvaliteto izvorne kode projekta. Z orodjem NDepend lahko analiziramo programsko kodo pisano v jezikih C#, VB.NET, MC++ in C++/CLI.

V tabeli 3.4 so prikazane metrike, katerih izračun orodje omogoča.

METRIKE, KI JIH ORODJE OMOGOČA		
Vrste metrik	Metrike	
Preštevanje vrstic izvorne kode	Štetje vrstic z izvorno kodo *LOC*	
	Štetje vrstic kode s komentarji *CLOC*	
Ostale metrike preštevanja	Izračun odstotka deleža komentarjev glede na celotno izvorno kodo	
	Število inštrukcij vmesniškega jezika *IL* (angl. *intermediate language*)	
	Število sklopov	
	Število imenskih prostorov	
	Število tipov	
	Število polj	
	Število metod	
	Število parametrov	
	Število spremenljivk	
	Število dohodnih odvisnosti *CA* (angl. *afferent coupling*)	
	Število odhodne odvisnosti *CE* (angl. *efferent coupling*)	
	Število otrok razreda *NOC*	
	Globina dedovanja *(*DIT)	
Računske metrike	McCabeova ciklomatična metrika	
	Vzdrževalski indeks *MI*	
	Preostale računske metrike	nestabilnost *I*
		relacijska kohezija *H*
		abstraktnost *A*
		rang $R(A)$
		pomanjkanje kohezije metod *LOCM*
Preostale metode	Normalizirana oddaljenost od glavnega zaporedja *D*	
	Stopnja paketa	
	Velikost primerka	
	Vrednost asociacije med razredi ABC	

Tabela 3.4 Metrike, ki jih omogoča orodje NDepend.

Orodje NDepend je zmožno odvisnosti med razredi, imenskimi prostori (angl. *namespace*) in sklopi zaznati takoj, ko so ustvarjene, obenem pa razvijalcu pomaga odvisnosti razrešiti, kar zniža nadaljne stroške razvoja in vzdrževanja. Odvisnosti so prikazane v obliki grafa.

Orodje NDepend omogoča natančen vizualen pregled enot, na katere bo naša sprememba vplivala. Če imajo te enote nadaljne odvisnosti, nam program izpiše tudi te. S pomočjo tega razvijalec porabi manj časa za analiziranje in oblavladovanje

odvisnosti, poleg tega pa se zmanjša verjetnost, da bi spreminjanje kode vodilo v nepredvidene situacije.

Poleg standardnih primerjav inkrementalnega razvoja programske kode, ki jih zmorejo narediti repozitoriji (primerjava 2 verzij izvorne kode), NDepend naredi še naslednje:

- spremembe komentarjev in/ali izvorne kode,
- zaznavanje odstranjene, dodane ali spremenjene izvorne kode in komentarjev,
- zaznavanje sprememb in opozarjanje na nekompatibilnosti.

Orodje Ndepend pomaga skrbeti za nespremenljivost in čistost objektov:

- objekt je nespremenjen, če se mu stanje zatem, ko je bil ustvarjen ne spremeni,
- funkcija je čista, če ne spremeni stanja polj.

NDepend s pomočno CQLinq pogojev in pravil pomaga razvijalcu zagotoviti nespremenljivost in čistost razredov in metod. Uporabnost tega je najbolj očitna pri večnitnih procesih, kjer je spreminjanje stanj pogosto nevarno početje, in pri funkcionalnem programiranju, kjer velja, da se stanja ne spreminjajo.

Orodje NDepend pomaga spremljati spreminjanje izvorne programske opreme:

- število vrstic izvorne kode,
- kolikokrat je bila poizvedba LINQ (angl. *language-integrated query*) prekršena in koliko poizvedb je programer prekršil; LINQ je funkcionalnost razvojnega okolja Visual Studio, s pomočjo katere izdelamo poizvedbe, s katerimi preverjamo ali pridobimo podatke in objekte;
- statistika o obsegu testirane kode,
- povprečne in najvišje vrednosti za različne metrike kvalitete programske kode,
- uporaba „third-party“ programske opreme,
- s pomočjo CQLinq poizvedb lahko izdelamo tudi svoje metrike spremljanja trendov in s tem razširimo funkcionalnost programa.

Kompleksnosti programske kode NDepend izpiše na naslednje načine:

- z drevesno strukturo metrike,

- z grafom trendov,
- s tabelo odvisnosti,
- z grafom odvisnosti,
- z grafom abstrakcije in nestabilnosti.

Poleg navedenih funkcionalnosti omenimo, da NDepend omogoča še naslednje:

- Kontinuirana poročila na podlagi integracije v razvojno okolje:
  - status razvoja,
  - na podlagi nastavljenih CQLing pravil lahko orodje NDepend razvijalca opozori na vsako kršenje nastavljenih CQLinq pravil.
- Preverjanje kvalitete prevajanja:
  - Težave pri različnih verzijah sklopov:
    - sklop A se sklicuje na sklop B verzije 2.1, na voljo pa je samo verzija 2.0,
    - sklop A se sklicuje na sklop B verzije 2.0, 2.1 itd.
  - Zaznavanje konfliktov sklopov:
    - v primeru, ko se sklop A sklicuje na sklop B, se lahko zgodi, da obstaja še nek drug sklop z istim imenom,
    - razlike med fizičnimi imeni sklopov in logičnimi imeni sklopov.
  - Zaznavanje problemov pri PDB datotekah (angl. *Program Database File*)
    - manjkajoče PDB datoteke,
    - PDB datoteke niso ustrezno sinhronizirane z izvorno kodo ali sklopi.

3.5 JArhitect

Pregledali smo še orodje JArhitect, ki je zelo podoben orodju NDepend, zato podrobnejši opis ni potreben. JArhitect je namenjen testiranju Jave, medtem ko je bil NDepend namenjen testiranju programov pisanih v jezikih C#, VB.NET, MC++, C++/CLI. Razlika je tudi v tem, da se NDepend lahko požene znotraj Visual Studia, medtem ko moramo JArhitect pognati samostojno.

3.6 Visual Studio

Za konec tega poglavja omenimo še to, da razvojno okolje Visual Studio prav tako omogoča analiziranje kompleksnosti izvorne programske kode. Namenjeno je analiziranju programov pisanih v .NET jezikih, torej C#, C++/CLI, ClojureCLR, Visual Basic .NET, itd.

V tabeli 3.5 se nahaja seznam metrik, ki jih zna izračunati razvojno okolje Visual Studio.

METRIKE, KI JIH ORODJE OMOGOČA	
Vrste metrik	Metrike
Preštevanje vrstic izvorne kode	Štetje vrstic z izvorno kodo *LOC*
	Globina dedovanja *(*DIT)
Računske metrike	McCabeova ciklomatična metrika
	Vzdrževalski indeks *MI*

Tabela 3.5 Metrike, ki jih omogoča razvojno okolje Visual Studio.

4 Zgledi avtomatiziranega pridobivanja metrik

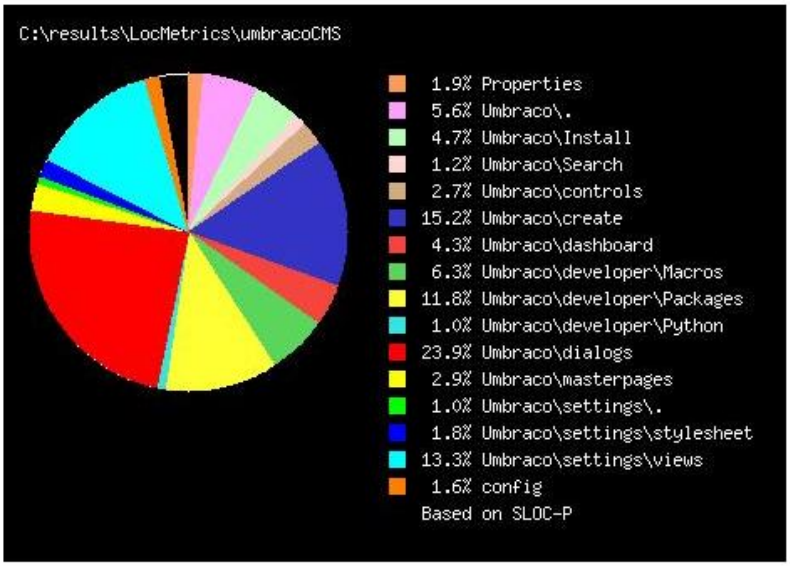
V pričujočem poglavju si bomo ogledali rezultate analiz izvorne kode programov Newtonsoft Json.NET, SignalR in UmbracoCMS. Te programe smo izbrali na Githubu, ki predstavlja sistem za nadzor različic izvorne programske kode. V prvem delu poglavja se bomo osredotočili na analiziranje manjšega dela programa UmbracoCMS in sicer del Web.UI. To bomo storili zato, da bomo lahko poleg orodij Locmetrics, CMT++, NDepend in analize v Visual Studiu uporabili tudi orodje Semantic Designs CSharp Software Metrics. Poskusna licenca za uporabo je bila pri tem orodju žal omejena na največ 1500 vrstic izvorne kode na datoteko ali največ 7500 vrstic izvorne kode v vseh izbranih datotekah. Nekatera programska orodja za analiziranje izvorne programske kode so bila brezplačna, druga pa so za uporabo zahtevala vsaj poskusno licenco. Ta je pri dveh orodjih omogočala neomejeno analiziranje, pri enem pa je bila, kot smo že omenili, žal omejena. Na koncu prvega dela poglavja bomo analizirali rezultate analiz različnih orodij, kjer bomo videli ali različna orodja metrike računajo na enak način. V drugem delu poglavja pa se bomo osredotočili na orodje NDepend in z njim analizirali celotno programsko kodo programov Newtonsoft Json.NET, SignalR in UmbracoCMS.

Program Newtonsoft Json.NET je priljubljeno ogrodje za delo z JSONom za .NET. Omogoča serializacijo za pretvarjanje med .NET objekti in JSONom, ki je hitrejša od v .NET vgrajene serializacije. Možno je tudi pretvoriti JSON v XML in obratno. ASP.NET SignalR je knjižnica, ki omogoča spletnim aplikacijam enostavno dodajanje funkcij, ki se izvajajo v realnem času. To sicer omogoča tudi ASP.NET MVC (angl. *model view controller*), vendar moramo pri uporabi MVCja spletno stran osvežiti, da se prikažejo spremembe. UmbracoCMS je priljubljen odprtokoden sistem za upravljanje vsebin (angl. *content management system*). Prednost UmbracoCMS je v brezplačnosti, poleg tega pa je prijazen programerjem. Podpira tako ASP.NET MVC, kot tudi ASP.NET Web Forms.

4.1 LocMetrics

Poročilo analize vsebuje število izvornih datotek, seštevek vrstic izvorne kode, seštevek pojavitev ključnih besed in seštevek komentarjev, kar nam da vpogled v osnovno strukturo izvorne kode. Izračun McCabeove ciklomatične kompleksnosti je predstavljen le kot seštevek vrednosti $v(G)$ vseh funkcij. Ta vrednost nam lahko služi za izračun gostote odločitev v programu, žal pa ne dobimo podatka o $v(G)$ na funkcijo. V rezultate je vključen tudi tortni graf, kjer je predstavljeno razmerje vrstic izvorne kode med izbranimi datotekami. Rezultate analize lahko orodje izpiše v HTML obliki ali v Excel datoteki. Orodje smo uporabili za analizo programa Newtonsoft Json.NET, SignalR in UmbracoCMS, za primer prikaza rezultatov pa bo zadoščal del programa UmbracoCMS, katerega poročilo analize je prikazano na slikah 4.1, 4.2 in 4.3. Tekom raziskovanja poročila se je hitro izkazalo, da medtem ko je orodje LocMetrics dobro pri analiziranju posamezne datoteke, se pri analizi večih datotek hitro pokaže pomanjkljiva preglednost poročila.

Na sliki 4.1 vidimo, da je orodje na podlagi števila fizičnih vrstic izvorne programske kode izračunalo, da je največji imenski prostor "dialogs".



Slika 4.1 Prikaz grafa iz poročila analize izvorne kode programa UmbracoCMS narejenega z orodjem LocMetrics. Graf prikazuje razmerje velikosti imenskih prostorov.

Na sliki 4.2 so predstavljeni rezultati preštevalnih metrik ter McCabeova ciklomatična metrika. V poročilu vidimo, da ima analizirani imenski prostor skupno 3371 vrstic izvorne kode in 1213 vrstic s komentarji, kar pomeni, da je koda verjetno dobro dokumentirana in bo nadgrajevanje tega dela kode v prihodnosti lažje.

Overall		
Symbol	**Count**	**Definition**
Source Files	66	Source Files
Directories	279	Directories
LOC	3371	Lines of Code
BLOC	509	Blank Lines of Code
SLOC-P	1649	Physical Executable Lines of Code
SLOC-L	1016	Logical Executable Lines of Code
MVG	97	McCabe VG Complexity
C&SLOC	34	Code and Comment Lines of Code
CLOC	1213	Comment Only Lines of Code
CWORD	4994	Commentary Words
HCLOC	8	Header Comment Lines of Code
HCWORD	60	Header Commentary Words

Slika 4.2 Prikaz poročila analize izvorne kode programa UmbracoCMS narejenega z orodjem LocMetrics. Poročilo prikazuje seštevke metrik za vse izbrane datoteke.

C:\results\LocMetrics\umbracoCMS - FOLDERS

Folder	Files	LOC	SLOC Physical	SLOC Logical	MVG	BLOC	C&SLOC	CLOC	CWORD	HCLOC	HCWORD
Total	66	3371	1649	1016	97	509	34	1213	4994	8	60
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI	66	3371	1649	1016	97	509	34	1213	4994	8	60
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Properties	2	61	31	12	2	12	1	18	110	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco	62	3260	1592	989	94	488	32	1180	4821	8	58
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\Install	2	167	78	49	2	31	1	58	230	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\Install\Legacy	2	167	78	49	2	31	1	58	230	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\Search	2	33	19	10	0	7	1	7	28	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\controls	4	122	45	29	0	19	2	58	221	0	2
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\controls\Images	2	24	12	5	0	5	1	7	28	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\create	10	555	250	146	13	84	7	221	900	0	5
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\dashboard	4	221	71	43	1	31	2	119	446	0	2
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\developer	8	574	315	205	26	85	4	174	757	0	4
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\developer\Macros	2	189	104	62	11	22	1	63	321	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\developer\Packages	4	349	195	134	15	57	2	97	382	0	2
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\developer\Python	2	36	16	9	0	6	1	14	54	0	1
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\dialogs	10	768	394	247	35	103	4	271	1161	8	32
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\masterpages	6	99	47	26	0	17	3	35	136	0	3
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\settings	8	503	265	171	15	80	4	158	639	0	4
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\settings\stylesheet	4	54	30	16	0	10	2	14	56	0	2
C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Umbraco\settings\stylesheet\property	2	27	15	8	0	5	1	7	28	0	1

Slika 4.3 Prikaz poročila analize izvorne kode programa UmbracoCMS narejenega z orodjem LocMetrics. Rezultati so predstavljeni po izbranih mapah.

4.2 Uporaba orodja Semantic Designs CSharp Software Metrics

V poročilu je zapisano število vrstic izvorne kode, število vrstic s komentarji, število datotek, število funkcij, vrednosti ciklomatične kompleksnosti $v(G)$, število odločitev in na podlagi tega tudi gostota odločitev, globina odvisnosti, vrednosti Halsteadove metrike in vzdrževalski indeks. Žal je bilo mogoče za testiranje pridobiti le poskusno verzijo programa, ki omogoča le do 1500 vrstic kode na datoteko in največ 7500 vrstic kode na vse testirane datoteke, zato smo izvedli analizo na imenskem prostoru Web.UI programa UmbracoCMS.

Listing 4.1 prikazuje analize izvorne kode vseh izbranih datotek programa Umbraco-CMS. Iz poročila izvemo, da ima projekt 3338 vrstic izvorne kode od tega 1587 vrstic z izvorno kodo napisano v programskem jeziku C#. Izvemo tudi, da je gostota odločitev

3.63, povprečna ciklomatična kompleksnost metod v vseh izbranih datotekah pa 2.13. Izpisane so tudi vrednosti Halsteadovih metrik za vse izbrane datoteke.

Listing 4.1 Poročilo vseh izbranih datotek.

```
                                        CSharp Software
                                            Metrics Summary
                                        Semantic Designs, Inc.
                                        http://www.semdesigns.
                                        com

Summary:
    3338 lines of source.
    1587 lines of CSharp code.
    1279 lines of comment.
    66 files.
    67 types.
    84 methods.
    Aggregate cyclomatic complexity: 179
    Mean cyclomatic complexity: 2.13
    Median cyclomatic complexity: 1.00
    Conditional statements: 100
    Decision density: 3.63
    Max loop depth: 1
    Max loop depth position: <file
        C:/source_code/Umbraco-CMS-dev-v7/src/Umbraco.Web.UI/
    Umbraco/create/
    PartialView.ascx.cs>::%global_declaration_space%.Umbraco.Web.
    UI.Umbraco.Create.PartialView @ line 39
    Max conditional nesting depth: 4
    Max conditional nesting depth position: <file
        C:/source_code/Umbraco-CMS-dev-v7/src/Umbraco.Web.UI/
    Umbraco/
    developer/Macros/EditMacro.aspx.cs>::%global_declaration_
    space%.Umbraco.Web.UI.Umbraco.Developer.Macros.EditMacro @
```

```
    line 72
Halstead unique operators: 648
Halstead unique operands: 2708
Halstead operator occurrence: 4808
Halstead operand occurrence: 5451
Halstead program length: 10259
Halstead program vocabulary: 3356
Halstead program volume: 120158.81
Halstead program difficulty: 652.19
Halstead program effort: 78366087.92
Halstead bug prediction: 40.05
SEI maintainability index: 130.59
```

Listing 4.2 prikazuje poročilo analize izvorne kode posamezne datoteke programa UmbracoCMS. Izpisani so isti podatki kot za vse izbrane datoteke, le da so tu izračunane vrednosti samo za metode v tej datoteki.

Listing 4.2 Poročilo posamezne datoteke.

```
FILE C:/source_code/Umbraco-CMS-dev-v7/src/Umbraco.Web.UI/
Properties/AssemblyInfo.cs
            25 lines of source.
            9 lines of CSharp code.
            10 lines of comment.
            0 types.
            0 methods.
            Aggregate cyclomatic complexity: 0
            Mean cyclomatic complexity: 0.00
            Median cyclomatic complexity: 0.00
            Conditional Statements: 0
            Decision density: 0.00
            Max loop depth: 0
            Max conditional nesting depth: 0
            Halstead unique operators: 7
            Halstead unique operands: 27
```

```
Halstead operator occurrence: 35
Halstead operand occurrence: 39
Halstead program length: 74
Halstead program vocabulary: 34
Halstead program volume: 0.00
Halstead program difficulty: 0.00
Halstead program effort: 0.00
Halstead bug prediction: 0.00
SEI maintainability index: 0.00
```

Listing 4.3 prikazuje poročilo analize izvorne kode posamezne metode programa UmbracoCMS. Izpisani so isti podatki kot za vse izbrane datoteke ali za posamezno izbrano datoteko, le da so tu izračunane vrednosti samo za eno metodo.

Listing 4.3 Poročilo posameznega razreda in metode.

```
CLASS Settings @ line 14
                19 lines of CSharp code.
                0 lines of comment.
                2 methods.
                Aggregate cyclomatic complexity: 2
                Mean cyclomatic complexity: 1.00
                Median cyclomatic complexity: 1.00
                Conditional statements: 0
                Decision density: 0.11
                Max loop depth: 0
                Max conditional nesting depth: 0
                Halstead unique operators: 13
                Halstead unique operands: 42
                Halstead operator occurrence: 61
                Halstead operand occurrence: 75
                Halstead program length: 136
                Halstead program vocabulary: 55
                Halstead program volume: 786.26
                Halstead program difficulty: 11.61
```

```
Halstead program effort: 9126.29
Halstead bug prediction: 0.26
SEI maintainability index: 108.86

METHOD Default @ line 20
    5 lines of CSharp code.
    0 lines of comment.
    Cyclomatic complexity: 1
    Conditional statements: 0
    Decision density: 0.20
    Max loop depth: 0
    Max conditional nesting depth: 0
    Halstead unique operators: 4
    Halstead unique operands: 3
    Halstead operator occurrence: 5
    Halstead operand occurrence: 5
    Halstead program length: 10
    Halstead program vocabulary: 7
    Halstead program volume: 28.07
    Halstead program difficulty: 3.33
    Halstead program effort: 93.58
    Halstead bug prediction: 0.01
    SEI maintainability index: 127.36
```

4.3 Uporaba orodja Testwell CMT++

V poročilu je za vsako datoteko in metodo priložen podatek o vrednosti ciklomatične komplesnosti $v(G)$, številu fizičnih vrstic izvorne kode, številu vrstic programske kode, Halsteadovi metriki volumna, Halsteadovi metriki ocene števila napak in vzdrževalski indeks. Sledi še povzetek metrik za vse izbrane datoteke, kjer so vse prej omenjene vrednosti seštete. Slednje lahko služi kot mera za kompleksnost testiranega programa.

Pri izvozu v tekstovno datoteko gre za kratko obliko poročila, kjer so zabeleženi podatki o tem, kdaj je bila analiza narejena in katere opcije so bile izbrane ob izvajanju

analize ter večina na začetku razdelka omenjenih metrik. Primer takšnega poročila prikazuje slika 4.4.

```
*******************************************************************************
*        CMT++, Complexity Measurement Tool for C/C++/C#, Version 6.0         *
*                                                                             *
*                          COMPLEXITY MEASURES REPORT                         *
*                                                                             *
*                     Copyright (c) 1993-2013 Testwell Oy                     *
*             Copyright (c) 2013-2015 Verifysoft Technology GmbH              *
*******************************************************************************

License notice: This is a limited period evaluation copy license.

This report was produced at Tue Feb 23 21:33:19 2016
Options: -o C:\results\Testwell\CMT\umbracoCMS\TEXT\report.txt -f C:\Users\Matjaz\cmt\files.txt


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\config\splashes\NoNodes.aspx.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================
  12 NoNodes::
  12   OnInit()                           2     11      8      143  0.02   128

  24 NoNodes.aspx.cs                      2     24     20  -   461  0.09   103
================================================================================


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\config\splashes\NoNodes.aspx.designer.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================

  24 NoNodes.aspx.designer.cs             1     24      6      136  0.02   141
================================================================================


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Properties\AssemblyInfo.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================

  24 AssemblyInfo.cs                      1     24     10      322  0.05   131
================================================================================


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Properties\Settings.Designer.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================
  21 Settings::
  21   get()                              1      3      3-      12- 0.00   140
  30   get()                              1      3      3-      35  0.01   135

  35 Settings.Designer.cs                 1     35     21  -   705  0.16   129
================================================================================
```

Slika 4.4 Prikaz rezultatov analize izvorne kode programa UmbracoCMS izvoženih v tekstovno datoteko.

Primeru tekstovnega izpisa sledi primer izvoza v HTML datoteko. Prav tako kot pri tekstovni obliki gre tudi tu za krajši povzetek. V splošnem pogledu HTML datoteke vidimo povzetek poročila, kjer izvemo, kdaj je bilo poročilo ustvarjeno in katere opcije so bile izbrane ob izvajanju analize in v tretjem poglavju omenjene metrike (glej 3.3). Najopaznejša razlika med HTML in tekstovno obliko je v preglednosti poročila, saj je oblikovano hiearhično glede na datoteke in funkcije ter glede na povzetek ter podrobno analizo, poleg tega pa HTML poročilo vključuje tudi kopijo testirane izvorne kode. Primer takšnega poročila prikazuje slika 4.5.

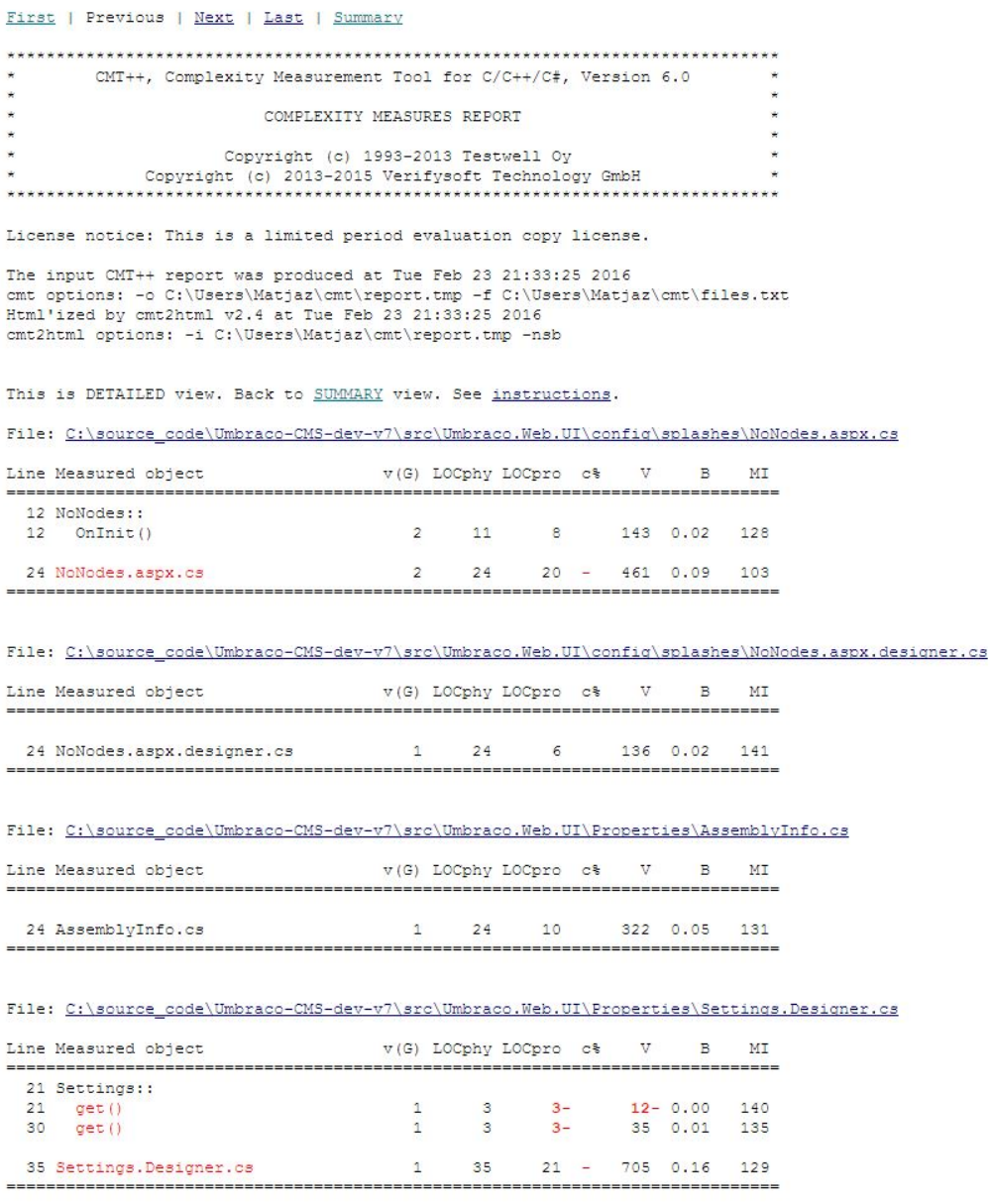
```
*************************************************************************
*        CMT++, Complexity Measurement Tool for C/C++/C#, Version 6.0   *
*                                                                       *
*                         COMPLEXITY MEASURES REPORT                    *
*                                                                       *
*                    Copyright (c) 1993-2013 Testwell Oy                *
*             Copyright (c) 2013-2015 Verifysoft Technology GmbH        *
*************************************************************************

License notice: This is a limited period evaluation copy license.

The input CMT++ report was produced at Tue Feb 23 21:33:25 2016
cmt options: -o C:\Users\Matjaz\cmt\report.tmp -f C:\Users\Matjaz\cmt\files.txt
Html'ized by cmt2html v2.4 at Tue Feb 23 21:33:25 2016
cmt2html options: -i C:\Users\Matjaz\cmt\report.tmp -nsb

This is SUMMARY view. Go to DETAILED view. See instructions.
```

Alarms-%	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
	NoNodes.aspx.cs	2	24	20	-	461	0.09	103
	NoNodes.aspx.designer.cs	1	24	6		136	0.02	141
	AssemblyInfo.cs	1	24	10		322	0.05	131
	Settings.Designer.cs	1	35	21	-	705	0.16	129
	ImageViewer.ascx.cs	1	7	7		134	0.03	114
	ImageViewer.ascx.designer.cs	1	15	5		70-	0.01	150
	PasswordChanger.ascx.cs	1	27	22	-	601	0.12	105
	PasswordChanger.ascx.designer.cs	1	69	11		491	0.09	118
	Create.aspx.cs	2	38	33	-	762	0.18	110
	Create.aspx.designer.cs	1	25	7		123	0.02	140
	DlrScripting.ascx.cs	1	10	10	-	214	0.04	106
	DlrScripting.ascx.designer.cs	1	15	5		61-	0.01	151
	PartialView.ascx.cs	4	66	53	-	1652	0.32	108
	PartialView.ascx.designer.cs	1	69	11		491	0.09	118
	PartialViewMacro.ascx.cs	8	81	65	-	2226	0.47	108
	PartialViewMacro.ascx.designer.cs	1	87	13		651	0.12	113
	User.ascx.cs	7	87	63	-	2327	0.52	121
	User.ascx.designer.cs	1	105	15		792	0.15	109
	xslt.ascx.cs	1	10	10	-	214	0.04	106
	xslt.ascx.designer.cs	1	15	5		61-	0.01	151
	ExamineManagement.ascx.cs	1	13	12	-	259	0.05	100
	ExamineManagement.ascx.designer.cs	1	60	10		395	0.08	122
	UserControlProxy.aspx.cs	3	39	34	-	774	0.19	92
	UserControlProxy.aspx.designer.cs	1	105	15		735	0.12	110
	EditMacro.aspx.cs	13	145	96	-	4409	0.95	119
	EditMacro.aspx.designer.cs	1	42	8		284	0.05	129
	DirectoryBrowser.aspx.cs	14	141	108	-	5578	1.36	101
	DirectoryBrowser.aspx.designer.cs	1	42	8		270	0.04	129
	StarterKits.aspx.cs	6	84	67	-	2247	0.48	85

Slika 4.5 Prikaz splošnega poročila analize izvorne kode programa UmbracoCMS izvoženega v HTML datoteko.

Sledi podrobno poročilo za vsako posamezno analizirano datoteko. V tem poročilu prav tako izvemo, kdaj je bilo narejeno in katere opcije so bile izbrane ob izvajanju analize ter v opisu orodja CMT++, predstavljene pa so tudi v tretjem poglavju omenjene metrike

za vse funkcije v datoteki. Poročilu je priložena tudi kopija izvorne kode, da je ob kasnejšem pregledovanju analize preglednost poročila kar najboljša. Primer takšnega poročila prikazuje slika 4.6.



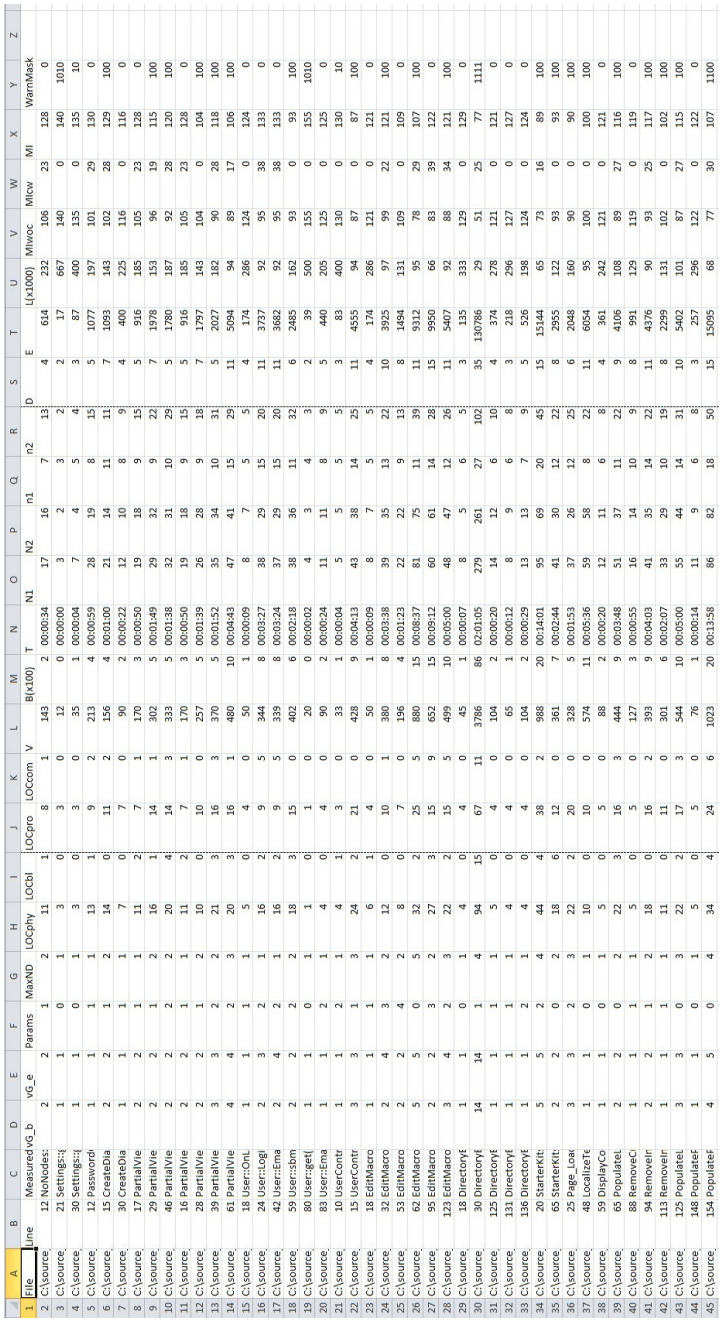
First | Previous | Next | Last | Summary

```
*******************************************************************************
*        CMT++, Complexity Measurement Tool for C/C++/C#, Version 6.0         *
*                                                                             *
*                          COMPLEXITY MEASURES REPORT                         *
*                                                                             *
*                    Copyright (c) 1993-2013 Testwell Oy                      *
*             Copyright (c) 2013-2015 Verifysoft Technology GmbH              *
*******************************************************************************

License notice: This is a limited period evaluation copy license.

The input CMT++ report was produced at Tue Feb 23 21:33:25 2016
cmt options: -o C:\Users\Matjaz\cmt\report.tmp -f C:\Users\Matjaz\cmt\files.txt
Html'ized by cmt2html v2.4 at Tue Feb 23 21:33:25 2016
cmt2html options: -i C:\Users\Matjaz\cmt\report.tmp -nsb


This is DETAILED view. Back to SUMMARY view. See instructions.

File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\config\splashes\NoNodes.aspx.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================
  12 NoNodes::
  12   OnInit()                           2     11      8      143  0.02   128

  24 NoNodes.aspx.cs                      2     24     20  -   461  0.09   103
================================================================================


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\config\splashes\NoNodes.aspx.designer.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================

  24 NoNodes.aspx.designer.cs             1     24      6      136  0.02   141
================================================================================


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Properties\AssemblyInfo.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================

  24 AssemblyInfo.cs                      1     24     10      322  0.05   131
================================================================================


File: C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\Properties\Settings.Designer.cs

Line Measured object                  v(G) LOCphy LOCpro  c%    V     B    MI
================================================================================
  21 Settings::
  21   get()                              1      3      3-      12- 0.00   140
  30   get()                              1      3      3-      35  0.01   135

  35 Settings.Designer.cs                 1     35     21  -   705  0.16   129
================================================================================
```

Slika 4.6 Prikaz podrobnega poročila analize izvorne kode programa UmbracoCMS narejene z orodjem CMT++.

Sledi primer izvoza poročila v excelovo datoteko. Tako kot tekstovna in HTML oblika je

tudi ta oblika izpisa krajši povzetek poročila. V tem poročilu so rezultati strukturirani glede na datoteko in metodo, zabeležene pa so iste metrike kot v HTML ali tekstovni obliki. Primer takšnega poročila prikazuje slika 4.7.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	File	Line	Measured vG_b		vG_e	Params	MaxND	LOCphy	LOCbl	LOCpro	LOCcom	V	B(x100)	T	N1	N2	n1	n2	D	E	L(x1000)	Mlvoc	Mlcw	MI	WarnMask	
2	C:\source		12 NoNodes:		2	1	2	11	1	8	1	143	2	00:00:34	17	16	7	13	4	614	232	106	23	128	0	
3	C:\source		21 Setting::i		1	0	1	3	0	3	0	12	0	00:00:00	3	2	3	2	2	17	667	140	0	140	1010	
4	C:\source		30 Setting::i		1	0	1	3	0	3	0	35	1	00:00:04	7	4	5	4	3	87	400	135	0	135	10	
5	C:\source		12 Password		1	1	1	13	1	9	2	213	4	00:00:59	28	19	8	15	5	1077	197	101	29	130	0	
6	C:\source		15 CreateDia		2	2	1	2	14	0	7	156	4	00:01:00	21	14	11	11	7	1093	143	102	28	129	100	
7	C:\source		30 CreateDia		1	1	1	1	7	0	7	90	2	00:00:22	12	10	8	9	4	400	225	116	0	116	0	
8	C:\source		17 PartialVie		2	2	1	1	11	2	7	170	3	00:00:50	19	18	9	15	5	916	185	105	23	128	0	
9	C:\source		29 PartialVie		2	2	1	2	16	1	14	302	5	00:01:49	29	32	9	22	7	1978	153	96	19	115	100	
10	C:\source		46 PartialVie		2	2	2	2	20	4	14	333	5	00:01:38	32	31	10	29	5	1780	187	92	28	120	100	
11	C:\source		16 PartialVie		2	2	1	1	11	2	7	170	3	00:00:50	19	18	9	15	5	916	185	105	23	128	0	
12	C:\source		28 PartialVie		2	2	1	2	10	0	10	257	5	00:01:39	26	28	9	18	7	1797	143	104	0	104	100	
13	C:\source		39 PartialVie		3	3	2	2	21	3	16	370	5	00:01:52	35	34	10	31	5	2027	182	90	28	118	100	
14	C:\source		61 PartialVie		4	4	2	3	20	3	16	480	10	00:04:43	47	41	15	29	11	5094	94	89	17	106	100	
15	C:\source		18 User::Onl		1	1	1	1	5	0	4	50	1	00:00:09	8	7	5	5	4	174	286	124	0	124	0	
16	C:\source		24 User::Logi		2	3	2	1	16	2	9	344	8	00:03:27	38	29	15	20	11	3737	92	95	38	133	0	
17	C:\source		42 User::Ema		2	4	2	1	16	2	9	339	8	00:03:24	37	29	15	20	11	3682	92	95	38	133	0	
18	C:\source		59 User::Sbm		2	2	2	1	18	3	15	402	6	00:02:18	38	36	11	32	6	2485	162	93	0	93	100	
19	C:\source		80 User::getI		1	1	0	1	1	0	1	20	0	00:00:02	4	3	4	3	2	39	500	155	0	155	1010	
20	C:\source		83 User::Ema		1	1	2	1	4	0	4	90	2	00:00:24	11	11	8	9	5	440	205	125	0	125	0	
21	C:\source		10 UserContr		1	1	2	1	4	1	3	33	1	00:00:04	5	5	5	5	3	83	400	130	0	130	10	
22	C:\source		15 UserContr		3	3	1	3	24	2	21	428	9	00:04:13	43	38	14	25	11	4555	94	87	0	87	100	
23	C:\source		18 EditMacro		1	1	1	1	6	1	4	50	1	00:00:09	8	7	5	5	4	174	286	121	0	121	0	
24	C:\source		32 EditMacro		2	4	3	2	12	0	10	380	8	00:03:38	39	35	13	22	10	3925	97	99	22	121	100	
25	C:\source		53 EditMacro		2	2	4	2	8	0	7	196	4	00:01:23	22	22	9	13	8	1494	131	109	0	109	0	
26	C:\source		62 EditMacro		5	5	0	5	32	2	25	880	15	00:08:37	81	75	11	39	11	9312	95	78	29	107	100	
27	C:\source		95 EditMacro		2	2	3	2	27	3	15	652	15	00:09:12	60	61	14	28	15	9950	66	83	39	122	0	
28	C:\source		123 EditMacro		3	4	2	3	22	2	15	499	10	00:05:00	48	47	12	26	11	5407	92	88	34	121	100	
29	C:\source		18 Directory(		1	1	0	1	4	0	4	45	1	00:00:07	8	5	6	5	3	135	333	129	0	129	0	
30	C:\source		30 Directory(		14	14	1	4	94	15	67	3786	86	02:01:05	279	261	27	102	35	130786	29	51	25	77	1111	
31	C:\source		125 Directory(		1	1	1	1	5	0	4	104	2	00:00:20	14	12	6	10	4	374	278	121	0	121	0	
32	C:\source		131 Directory(		1	1	1	1	4	0	4	65	1	00:00:12	8	9	6	8	3	218	296	127	0	127	0	
33	C:\source		136 Directory(		1	1	2	1	4	0	4	104	2	00:00:29	13	13	7	9	5	526	198	124	0	124	0	
34	C:\source		20 StarterKit:		5	5	2	4	44	4	38	988	20	00:14:01	95	69	20	45	15	15144	65	73	16	89	100	
35	C:\source		65 StarterKit:		2	2	0	2	18	6	12	361	7	00:02:44	41	30	12	22	8	2955	122	93	0	93	100	
36	C:\source		25 Page_Leal		3	3	2	3	22	2	20	328	5	00:01:53	37	26	12	25	6	2048	160	90	0	90	100	
37	C:\source		48 LocalizeFe		1	1	0	1	10	0	10	574	11	00:05:36	59	58	8	22	11	6054	95	100	0	100	100	
38	C:\source		59 DisplayCo		1	1	0	1	5	0	5	88	2	00:00:20	12	11	6	8	4	361	242	121	0	121	0	
39	C:\source		65 Populate(		2	2	0	2	22	3	16	444	9	00:03:48	51	37	11	22	9	4106	108	89	27	116	100	
40	C:\source		88 RemoveCi		1	1	1	1	5	0	5	127	3	00:00:55	16	14	10	9	8	991	129	119	0	119	0	
41	C:\source		94 RemoveIr		2	2	1	2	18	0	16	393	9	00:04:03	41	35	14	22	11	4376	90	93	25	117	100	
42	C:\source		113 RemoveIr		1	1	1	1	11	0	11	301	6	00:02:07	33	29	10	19	8	2299	131	102	0	102	100	
43	C:\source		125 Populate(		3	3	0	3	22	2	17	544	10	00:05:00	55	44	14	31	10	5402	101	87	27	115	100	
44	C:\source		148 Populate(		1	1	0	1	5	0	5	76	1	00:00:14	11	9	6	8	3	257	296	122	0	122	0	
45	C:\source		154 Populate(		4	5	0	4	34	4	24	1023	20	00:13:58	86	82	18	50	15	15095	68	77	30	107	1100	

Slika 4.7 Prikaz rezultatov analize izvorne kode izvoženih v excelovo datoteko.

Na koncu si oglejmo še obsežni izpis v XML datoteko. Gre za način izpisa, ki nam prikaže vse, kar je program CMT++ zmožen izmeriti. Osnovni namen takšnega poročila je nadaljna uporaba s programi podjetja Testwell, lahko pa sami izdelamo program in podatke uvozimo iz dobljenih datotek ter jih obdelamo po lastnih željah. Primer poročila za uvoz in obdelavo prikazuje slika 4.8.

```
<file name="C:\source_code\Umbraco-CMS-dev-v7\src\Umbraco.Web.UI\config\splashes\NoNodes.aspx.cs" stamp="1455640358">
    <function name="NoNodes::OnInit()">
        <start_line>12</start_line>
        <vG_b>2</vG_b>
        <vG_e>2</vG_e>
        <params>1</params>
        <LOCphy>11</LOCphy>
        <LOCpro>8</LOCpro>
        <LOCbl>1</LOCbl>
        <LOCcom>1</LOCcom>
        <N>33</N>
        <N1>17</N1>
        <N2>16</N2>
        <n>20</n>
        <n1>7</n1>
        <n2>13</n2>
        <V>142.624</V>
        <B>0.024</B>
        <D>4.308</D>
        <E>614.379</E>
        <L>0.232</L>
        <T>00:00:34</T>
        <MaxND>2</MaxND>
        <MIwoc>106</MIwoc>
        <MIcwc>23</MIcwc>
        <MI>128</MI>
        <operators>
            <token count="4">()</token>
            <token count="5">.</token>
            <token count="3">;</token>
            <token count="1">=</token>
            <token count="1">base</token>
            <token count="1">if()</token>
            <token count="2">{}</token>
        </operators>
        <operands>
            <token count="1">&amp;quot;~/&amp;quot;</token>
            <token count="1">ContentCache</token>
            <token count="1">Current</token>
            <token count="1">EventArgs</token>
            <token count="1">HasContent</token>
            <token count="2">OnInit</token>
            <token count="1">Redirect</token>
            <token count="1">Response</token>
            <token count="1">UmbracoContext</token>
            <token count="2">e</token>
            <token count="2">store</token>
            <token count="1">var</token>
            <token count="1">void</token>
        </operands>
    </function>
    <file_total>
        <vG_b>2</vG_b>
        <vG_e>2</vG_e>
        <vG_b_max>2</vG_b_max>
        <vG_b_avg>2</vG_b_avg>
        <vG_e_max>2</vG_e_max>
        <vG_e_avg>2</vG_e_avg>
        <params>1</params>
        <LOCphy>24</LOCphy>
        <LOCpro>20</LOCpro>
        <LOCbl>4</LOCbl>
        <LOCcom alarmed="1">1</LOCcom>
        <N>86</N>
```

Slika 4.8 Prikaz rezultatov analize izvorne kode izvoženih v XML datoteko.

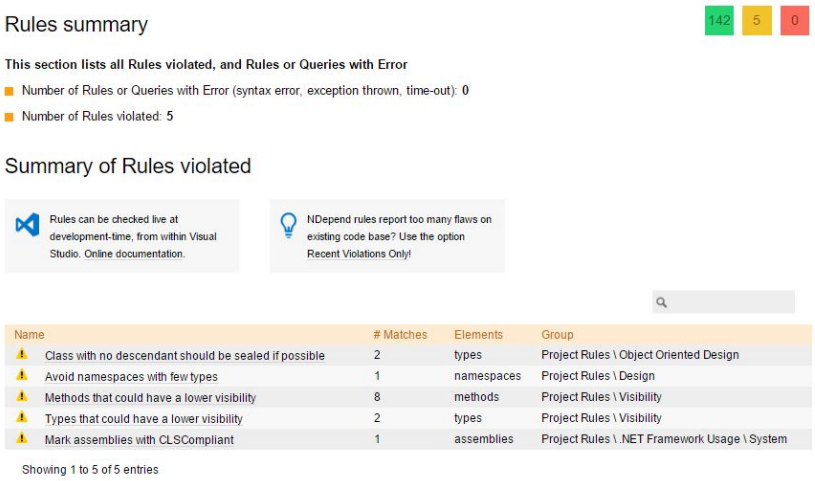
4.4 Uporaba orodja NDepend

V poročilu analize so podatki o tem kdaj je bila analiza narejena, koliko časa je trajala, katero verzijo programa in s katero verzijo orodja smo ta program testirali. Poročilo zajema diagrame in tabelo odvisnosti, drevesni pogled ciklomatične kompleksnosti, graf abstraktnosti in stabilnosti programa ter natančne podatke o izvorni kodi. Slednje zajema število sklopov, imenskih prostorov, metod, polj in datotek z izvorno kodo. Dobimo tudi podatek o številu komentarjev in deležu teh glede na celotno število vrstic izvorne kode. Pridobimo tudi podatke o tuji izvorni kodi, ki jo naš program potrebuje. V nadaljevanju si oglejmo primere poročil, ki jih naredi orodje NDepend.

Slika 4.9 prikazuje izpis metrik za celoten testiran projekt ali rešitev. Vidimo koliko vrstic izvorne kode ima projekt, koliko različnih tipov ima (sklopov, imenskih prostorov, metod, polj in datotek z izvorno kodo), koliko komentarjev ima glede na preostalo kodo, kolikšna je najvišja dosežena in kolikšna povprečna ciklomatična kompleksnost in koliko „third-party“ programske opreme program uporablja.

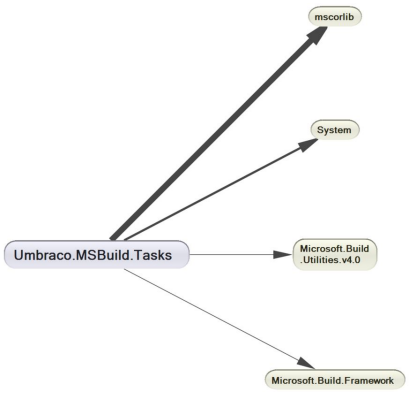
Slika 4.9 Prikaz splošnih rezultatov analize izvorne kode narejenih z orodjem NDepend.

Na sliki 4.10 je predstavljen izpis pravil, ki smo jih programirali s pomočjo LINQ (angl. *language-integrated query*). Izpis je sortiran glede na skupino pravil, pove pa nam število ponovitev kršitev določenega pravila. Imamo tudi možnost izločanja izpisov, tako da lahko vidimo le zadnje ali le kritične kršitve pravil.



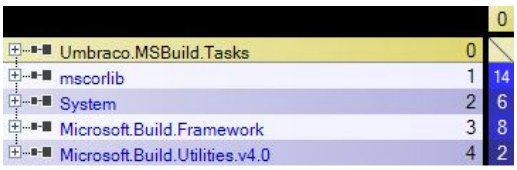
Slika 4.10 Prikaz rezultatov analize programiranih pravil pisanja izvorne kode narejenih z orodjem NDepend.

Na sliki 4.11 si lahko ogledamo graf odvisnosti. Ta graf nam hitro prikaže morebitne cikle odvisnosti.



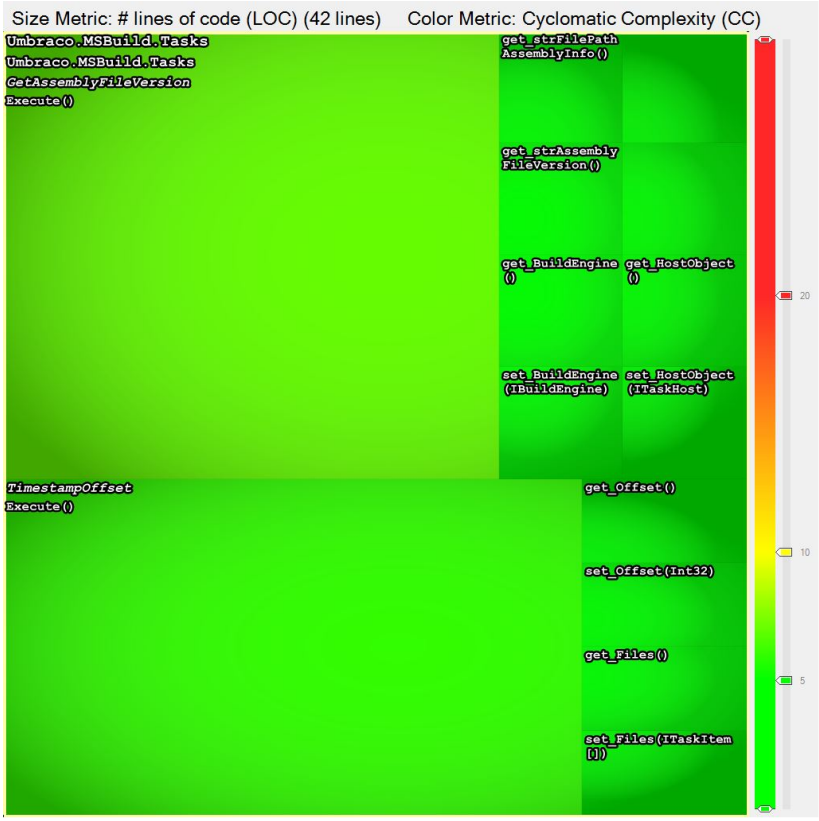
Slika 4.11 Prikaz grafa odvisnosti izvorne kode.

Na sliki 4.12 si lahko ogledamo tabelo odvisnosti. V tabeli hitro vidimo pomembnost določenih razredov na podlagi števila tipov, ki so odvisni od tega razreda, kar je izpisano na desni strani tabele. Iz tabele lahko na primer smatramo, da je razred „mscorlib“ v tem sklopu najpomembnejši.



Slika 4.12 Prikaz tabele odvisnosti izvorne kode.

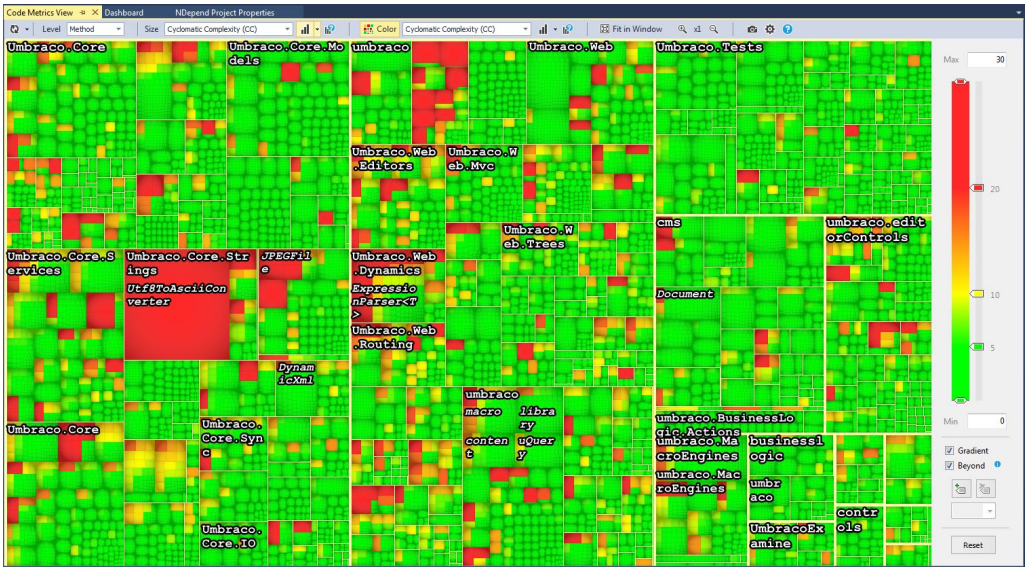
Na sliki 4.13 si lahko ogledamo drevesno strukturo prikaza ciklomatične kompleksnosti po različnih metodah. Metode na tej sliki so razvrščene glede na imenske prostore ter vrednost ciklomatične kompleksnosti. Na drevesni strukturi lahko vidimo, da je metoda Newtonsoft.Json.Serialization zelo kompleksna, medtem ko je metoda JsonTextWriter nekompleksna. Na podlagi tega lahko smatramo, da bo nadaljni razvoj imenskega prostora Web.UI potekal brez težav.



Slika 4.13 Prikaz ciklomatične kompleksnosti za metode razvrščene glede na imenske prostore ter vrednost ciklomatične kompleksnosti.

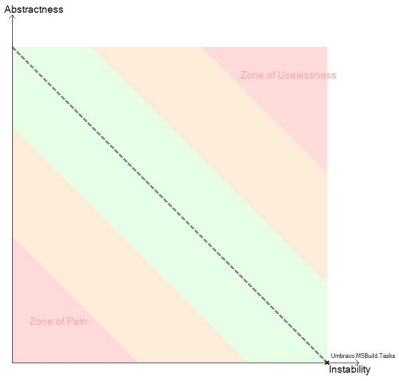
Vidimo lahko, da nam drevesna struktura pri analiziranju izvorne programske kode Um-

bracoCMS Web.UI ne prikaže uporabnosti prikaza v obliki drevesne strukture, zato si oglejmo še prikaz drevesne strukture McCabeove ciklomatične metrike za celoten program UmbracoCMS na sliki 4.14. Opazimo, da je najkompleksneji del programa UmbracoCMS razred Utf8ToAsciiConverter. Najkompleksnejša metoda v tem razredu je ToAscii z vrednostjo ciklomatične kompleksnosti 1313.



Slika 4.14 Prikaz ciklomatične kompleksnosti za celoten program UmbracoCMS, ki prikazuje metode razvrščene glede na imenske prostore ter vrednost ciklomatične kompleksnosti.

Na sliki 4.15 si lahko ogledamo graf abstraktnosti in stabilnosti za celoten imenski prostor. Na njem je prikazana težavnost razumevanja izvorne kode programa, stabilnost delovanja programa ter težavnost nadgrajevanja ter uporabnost programa. Iz grafa sklepamo, da je imenski prostor Web.UI programa UmbracoCMS neabstrakten in nestabilen.



Slika 4.15 Prikaz abstraktnosti in stabilnosti za celoten projekt. Idealen projekt bi se nahajal natanko na sredini grafa.

Na sliki 4.16 so izpisane statistike programa glede na metrike. Za vsako metriko izvemo, v kateri datoteki se nahaja najvišja izmerjena vrednost in povprečje. Iz poročila lahko takoj vidimo, katera metoda predstavlja največje tveganje zaradi visoke vrednosti ciklomatične kompleksnosti.

Application Statistics

Stat	# Occurences	Avg	StdDev	Max
Properties on interfaces	interfaces	0	0	-1 properties on
Methods on interfaces	interfaces	0	0	-1 methods on
Arguments on methods on interfaces	methods	0	0	-1 arguments on
Public properties on classes	2 Classes	3	1	4 public properties on Umbraco.MSBuild.Tasks.GetAssemblyFileVersion
Public methods on classes	2 classes	8	2	10 public methods on Umbraco.MSBuild.Tasks.GetAssemblyFileVersion
Arguments on public methods on classes	16 methods	0.38	0.48	1 arguments on Umbraco.MSBuild.Tasks.GetAssemblyFileVersion.set_strFilePathAssemblyInfo(String)
IL instructions in non-abstract methods	16 methods	14.69	29.46	97 IL instructions in Umbraco.MSBuild.Tasks.GetAssemblyFileVersion.Execute()
Cyclomatic complexity on non abstract Methods	16 Methods	2.06	2.86	CC = 11 for Umbraco.MSBuild.Tasks.GetAssemblyFileVersion.Execute()

Slika 4.16 Prikaz statistik programa.

Na sliki 4.17 so izpisani rezultati metrik za preverjanje kvalitete izvorne programske kode.

Types Metrics : Code Quality

Type Name	Type Rank	# Lines Of Code	# IL Instructions	# Lines Of Comment	% Comment	Cyclomatic Complexity	IL Cyclomatic Complexity	% Coverage	Afferent Coupling
GetAssemblyFileVersion	0.15	24	129	0	0	15	20	-	0
TimestampOffset	0.15	18	106	0	0	10	13	-	0

Showing 1 to 2 of 2 entries

Slika 4.17 Prikaz metrik za analizo kvalitete izvorne programske kode.

Na sliki 4.18 vidimo koliko instanc metod in koliko dedovanj se nahaja v tem delu kode.

Types Metrics : Code Members and Inheritance

Type Name	# Instance Methods	Nb Static Methods	Nb Properties	# Fields	# Children Classes	Depth Of Inheritance Tree	Type Namespace
GetAssemblyFileVersion	10	0	4	4	0	1	Umbraco.MSBuild.Tasks
TimestampOffset	6	0	2	2	0	2	Umbraco.MSBuild.Tasks

Showing 1 to 2 of 2 entries

Slika 4.18 Prikaz števila instanc metod ter dedovanja.

Na sliki 4.19 vidimo prikaz izračunov metrik za preverjanje kohezije med metodami in asociacij med razredi.

Types Metrics : Lack Of Cohesion Of Methods and Association Between Classes

Type Name	Lack Of Cohesion Of Methods	Lack Of Cohesion Of Methods HS	Association Between Classes	Type Namespace
GetAssemblyFileVersion	0	0	16	Umbraco.MSBuild.Tasks
TimestampOffset	0	0	10	Umbraco.MSBuild.Tasks

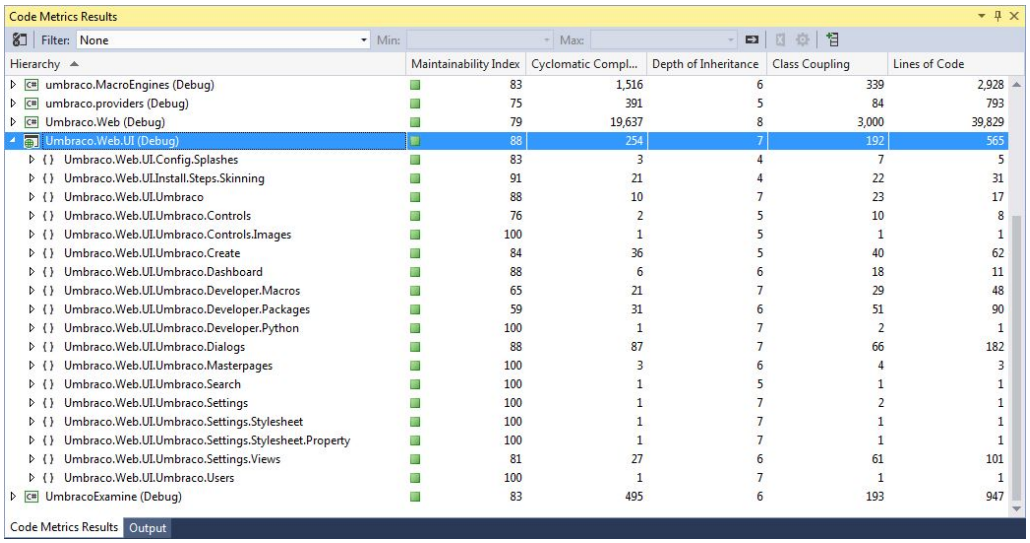
Showing 1 to 2 of 2 entries

Slika 4.19 Prikaz pomanjkanja kohezije med metodami in asociacij med razredi.

4.5 Visual Studio

V poročilu analize so vsebovani podatki o številu vrstic izvorne programske kode LOC, vrednosti McCabeove ciklomatične metrike, indeksu vzdrževalnosti MI in globini dedovanja. Podatki so v poročilu prikazani v obliki tabele in se jih lahko shrani v XML datoteko.

Na sliki 4.20 vidimo prikaz izračunov metrik za analiziranje kompleksnosti izvorne kode v razvojnem okolju Visual Studio.



Slika 4.20 Prikaz poročila analize kompleksnosti izvorne programske kode v razvojnem okolju Visual Studio.

4.6 Primerjava rezultatov

V tem razdelku primerjamo med seboj rezultate analiz orodij LocMetrics, Semantic Design CSharp Software Metrics, Testwell CMT++, NDepend in orodja vgrajenega v razvojno okolje Visual Studio za vse izbrane datoteke programa UmbracoCMS. Analiziramo, ali so rezultati med seboj primerljivi in ali orodja metrike interpretirajo na enak način. Pri preštevalnih metrikah bomo zagotovo opazili razlike pri seštevkih. S primerjavo se bomo osredotočili na preštevalne metrike *LOC*, *CLOC*, McCabeovo ciklomatično metriko in vzdrževalski indeks.

V tabeli 4.1 lahko takoj opazimo, da so razlike že pri osnovnem štetju vrstic z izvorno kodo *LOC*. Orodja namreč različno interpretirajo štetje vrstic programske kode. Pri

NDepend smo pri seštevku *LOC* venomer opazili rezultat 536, pri Visual Studiu pa 565 vrstic izvorne programske kode. Glede na to lahko sklepamo, da so preostala orodja analizirala vso izvorno programsko kodo v tej mapi, medtem ko se je Visual Studio omejil le na projekt Web.UI, temu primerna pa je potem tudi vrednost v orodju NDepend, ki je neposredno odvisen od Visual Studia. Pri štetju vrstic z izvorno programsko kodo *SLOC* opazimo, da je razlika med orodji manjša. Iz tega sklepamo, da so vsa orodja pravilno zaznala datoteke za test, razlika v številu *SLOC* pa se skriva v različni interpretaciji štetja vrstic kode. Nekatera orodja namreč *SLOC* smatrajo kot fizične vrstice kode, medtem ko nekatera *SLOC* smatrajo kot logične vrstice kode. LocMetrics in CMT++ štejeta enako, medtem ko SD CSharp Software Metrics šteje nekoliko drugače, NDepend pa tega rezultata ni sporočil. Pri štetju vrstic kode s komentarji *CLOC* opazimo, da so rezultati seštevkov zelo podobni, kljub temu pa se zopet pojavijo odstopanja. CMT++ in NDepend očitno vrstice kode s komentarji štejeta na enak način, medtem ko LocMetrics in SD CSharp Software Metrics štejeta nekoliko drugače.

Analizo nadaljujemo s primerjavo rezultatov McCabeove ciklomatične metrike. Izkaže se, da LocMetrics in CMT++ seštevek za vse funkcije naredita enako, medtem ko pri SD CSharp Software Metrics opazimo veliko odstopanje. NDepend je nudil odlične prikaze ciklomatične metrike v obliki grafov, žal pa ni nudil seštevka, ki bi ga lahko uporabili v primerjavi. Vrednost, ki jo je izračunalo razvojno okolje Visual Studio, je precej višja od preostalih izračunov, zato bi bilo potrebno te natančno preučiti in morda za razlago vprašati proizvajalce orodij, zakaj je do takšnega odstopanja prišlo. Izračun povprečja vrednosti McCabeove ciklomatične metrike na funkcijo nudita orodji SD CSharp Software Metrics in NDepend in izkaže se, da je zopet ne izračunata enako.

Na koncu si oglejmo še vzdrževalski indeks *MI*. Podatek za to sta nudili le orodjiSD CSharp Software Metrics in CMT++ ter razvojno okolje Visual Studio, pri izračunu pa se pojavi odstopanje.

Primerjava rezultatov analiz							
Vrste metrik	Metrike		LocMetrics	SD CSharp Software Metrics	Testwell CMT++	NDepend	Visual Studio
Preštevanje vrstic izvorne kode	Štetje vseh vrstic z izvorno kodo *LOC*		3371	3338	3305	536	565
	Štetje vrstic z izvorno programsko kodo *SLOC*		1649	1587	1649	/	/
	Štetje vrstic kode s komentarji *CLOC*		1213	1279	1247	1247	/
Računske metrike	McCabeova ciklomatična metrika	Seštevek vrednosti za vse metode	97	179	97	/	254
		Povprečna vrednost	/	2.13	/	2.34 /	/
	Vzdrževalski indeks MI	brez komentarjev	/	/	94	/	/
		teža komentarjev	/	/	23	/	/
		MI	/	130.59	94	/	88

Tabela 4.1 Rezultati analize

Za analizo z navedenimi orodji smo vsakič izbrali isto mapo z izvorno kodo, kljub temu pa je pri rezultatih prišlo do odstopanja. To se je zgodilo, ker nekatera orodja analizirajo izključno datoteke z izvorno kodo C#, medtem ko druga lahko analizirajo tudi datoteke z drugo izvorno kodo. LocMetrics lahko recimo analizira različne jezike hkrati in rezultate enostavno sešteje, medtem ko kompleksnejša orodja tega ne omogočajo. Medtem ko je LocMetrics analiziral programske jezike C#, C++, Java in SQL, je SD CSharp Software Metrics analiziral izključno programski jezik C#. Enako se je zgodilo pri orodju CMT++. Orodje NDepend je analiziralo datoteke s programsko kodo C#, VB.NET, MC++ in C++/CLI, a le tiste, ki so bile pred tem prevedene z razvojnim okoljem Visual Studio. Najverjetneje od tod izhaja razlika v izračunih, saj je razvojno okolje med gradnjo dodalo še nekatere zunanje vire, od katerih je bil testirani sklop odvisen, orodje pa je potem analiziralo ustvarjen sklop (angl. *assembly*)

Če bi želeli dobiti identične rezultate, bi morali kodo različnih programskih jezikov izločiti ter jo testirati posamično s primernim orodjem. CMT++ bi torej dodali orodje CMTJava, orodju NDepend pa bi dodali orodje JArhitect, nato pa rezultate znova primerjali med seboj.

4.7 Primerjava rezultatov analize celotnih programov

V tem razdelku smo z enim orodjem analizirali programe Newtonsoft Json.NET, SignalR in UmbracoCMS, rezultate pa prikazali v tabeli. S tem smo prikazali uporabno vrednost analiz kompleksnosti izvorne programske kode. Za namen te primerjave predpostavimo, da so programi Newtonsoft Json.NET, SignalR in UmbracoCMS dejansko program, ki smo ga naročili pri treh različnih zunanjih izvajalcih. Predpostavimo, da želimo preveriti, kakšna je kompleksnost dostavljene izvorne kode na tak način, da dobljene razultate lahko primerjamo med seboj. Odločili smo se uporabiti orodje CMT++, saj smo s tem orodjem lažje tabelarno prikazali pridobljene informacije, kljub temu da orodje NDepend nudi izračun večih metrik in poročilo prikaže na pregleden grafični način. Pregledali bomo obseg, kompleksnost in vzdrževalski indeks programa. Dobljene podatke predstavimo v tabeli 4.2.

Primerjava rezultatov analiz narejenih z orodjem CMT++					
Vrste metrik	Metrike		Newtonsoft Json.NET	SignalR	UmbracoCMS
Preštevanje vrstic izvorne kode	Štetje vseh vrstic z izvorno kodo *LOC*		55557	37620	415765
	Štetje vrstic z izvorno programsko kodo *SLOC*		37336	27379	281677
	Štetje vrstic kode s komentarji *CLOC*		11663	4893	82818
Računske metrike	McCabeova ciklomatična metrika	Seštevek vrednosti za vse metode	6735	2015	21838
		Povprečna vrednost	/	/	/
	Vzdrževalski indeks MI	brez komentarjev	98	104	100
		teža komentarjev	22	21	28
		MI	120	124	128

Tabela 4.2 Rezultati analize

5 Zaključek

V diplomski nalogi smo analizirali nekaj programskih metrik, ki so danes v rabi, poleg tega pa smo testirali različna orodja za avtomatizacijo ocene kompleksnosti izvorne programske kode na različno kompleksnih programih. Ugotovili smo, da pri izbiri ustreznega orodja igrata pomembno vlogo tako seznam metrik, ki jih orodje zna izračunati, kot tudi strukturiranost poročila, ki ga je orodje zmožno narediti. Ugotovili smo, da orodja za avtomatizacijo ocene kompleksnosti izvorne programske kode bistveno poenostavijo razvoj programske opreme tako, da nas sproti opozarjajo na morebitne težave pri razvoju in nam dajo oceno tveganja za vsak posamezen sklop programa. Na podlagi teh informacij se potem lahko odločimo, kako bomo ravnali v nadaljevanju. Če sklop v poročilu avtomatiziranih orodij ni videti preveč kompleksen, ga lahko popravimo, sicer bo ceneje in lažje ga razviti znova. To nam bistveno pomaga, če za razvoj različnih sklopov programske opreme najamemo zunanje izvajalce, saj ob prevzetju izdelka lahko programsko kodo analiziramo in tako dobimo boljšo predstavo o tem, kakšen izdelek imamo.

Izkaže se, da je avtomatizacija ocene kompleksnosti izvorne kode bistvena pri razvoju programske opreme, saj lahko neposredno vpliva na stroške nadaljnjega razvoja in kvalitete

izvorne programske kode, sploh če gre za dolgotrajen projekt, ki se ga bo v prihodnosti nadgrajevalo. Menim, da se zaradi pojavitve vedno bolj abstraktnih in kompleksnih programskih jezikov pojavlja vedno večja potreba po kvalitetnih orodjih za avtomatizacijo testiranja in ocenjevanja kompleksnosti izvorne programske kode, saj enostavno štetje vrstic kode ni več dovolj za pridobitev uporabnih informacij o izvorni kodi programa.

Literatura

[1] (2010) A. Abran, Software Metrics and Software Metrology.
Dostopno na: http://goo.gl/vmp8AV

[2] M. Andersson, Object-Oriented Design Quality Metrics, magistrsko delo thesis, Information Technology, Computing Science Department, Uppsala University, Uppsala, Sweden (2004).

[3] K. Bhatt, V. Tarey, P. Patel, K. B. MITS, D. Ujjain, Analysis Of Source Lines Of Code(SLOC) Metric, International Journal of Emerging Technology and Advanced Engineering (2) (2012).
Dostopno na: http://goo.gl/msIikn

[4] (2007) S. Celarier, S. Hanselman, P. Cauldwell, Ndepend metrics.
Dostopno na: http://goo.gl/AkYkwo

[5] (2006) J. Gorman, OO Design Principles and Metrics.
Dostopno na: http://goo.gl/AkYkwo

[6] J. W. Grenning, Unit Testing in C: Tools and Conventions (2013).
Dostopno na: http://www.drdobbs.com/testing/unit-testing-in-c-tools-and-conventions/240156344

[7] A. Kranjc, Model za vrednotenje pridobitev uporabe programskih tovarn pri razvoju informacijskih rešitev, magistrsko delo, Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, Maribor, Slovenija (2011).

[8] (2016) LocMetrics, LocMetrics.
Dostopno na: http://locmetrics.com/index.html

[9] A. Madi, O. K. Zein, S. Kadry, On the Improvement of Cyclomatic Complexity Metric, International Journal of Software Engineering and Its Applications, no. 2, (2013).

[10] T. McCabe, A. H. Watson, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, NIST Special Publication 500-235, (1996).

[11] H. Melton, E. Tempero, An Epirical Study of Cycles among Classes in Java, Auckland, New Zealand, 2007.
Dostopno na: https://goo.gl/IdaaQR

[12] MSDN, Namespace (2016).
Dostopno na: https://msdn.microsoft.com/sl-si/library/z2kcy19k.aspx

[13] V. Nguyen, S. D. Rubin, T. Tan, B. Boehm, A SLOC Counting Standard, COCOMO II Forum (2007).
Dostopno na: http://goo.gl/tivZFA

[14] (2011) A. Serebrenik, Software metrics (2).
Dostopno na: http://www.win.tue.nl/ aserebre/2IS55/2011-2012/10.pdf

[15] (2012) P. Smacchia, Cut off wrong dependencies in your .NET code.
Dostopno na: http://www.infoq.com/articles/NDepend

[16] (2012) Verifysoft, Measurement of Software Complexity with Testwell CMT++, Testwell CMTJava.
Dostopno na: http://goo.gl/y5c5iU

[17] (2016) Semantic Designs, Semantic Designs: Software Metrics Tools.
Dostopno na: http://www.semdesigns.com/Products/Metrics