

Verifikacija softvera — Statička analiza —

Milena Vujošević Janičić

www.matf.bg.ac.rs/~milena

Matematički fakultet, Univerzitet u Beogradu

Pregled

1 Pregledi koda

2 Automatska statička analiza

3 Literatura

Statička analiza

Statička analiza je ...

... analiza koda bez njegovog izvršavanja sa ciljem pronalaženja defekata u kodu.

Statička analiza može biti ...

... u obliku pregleda ili automatizovana.

Pregled

1 Pregledi koda

- Formalni pregledi
- Neformalni pregledi
- Uticaj pregleda
- Efikasno pregledanje

2 Automatska statička analiza

3 Literatura

Statička analiza — pregledi koda

Pregledi koda (engl. *code review*)

- Pregledi obuhvataju ljudske kontrole koda najčešće pre nego što kôd uđe u glavni repozitorijum
- Pregledima se otkrivaju razne vrste grešaka
- Cilj pregleda je povećanje kvaliteta koda, kako smanjivanjem broja grešaka, tako i po pitanju poštovanja pravila kodiranja i dokumentovanja koda
- Pregledima se ostvaruju i razni drugi ciljevi
- Pregledi ne mogu da garantuju da greške neće promaći

Šta se pregleda?

Pregledi daju odgovore na razna pitanja...

- Da li postoje neke očigledne logičke greške u kodu?
- Ako se posmatraju zahtevi koji su postavljeni, da li su svi slučajevi pokriveni i u potpunosti implementirani?
- Da li su novi testovi koji se dodaju dovoljni za nov kôd?
- Da li postojeći testovi treba da se promene da bi se uzele u obzir nove promene?
- Da li novi kod prati opšti stil programiranja na projektu?
- Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje (npr polimorfizam umesto if-ova, postojeće implementacije funkcija ili bibliotečke funkcije umesto novo-napisanih funkcija...)?

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje?

```
int x, y, Q, p, A, b, c, D, d;
scanf("%d%d%d%d",&x,&y,&p,&d);
Q = 2*(x + y);
c = x*y;
A = 2*(p+b);
D = p*b;
```

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje?

```
int x, y, Q, p, A, b, c, D, d;  
scanf("%d%d%d%d",&x,&y,&p,&d);  
Q = 2*(x + y);  
c = x*y;  
A = 2*(p+b);  
D = p*b;
```

- Potrebno je preimenovati sve promenljive. Imenovanje je nekonzistentno (velika i mala slova) i nedeskriptivno (nije jasan smisao promenljivih).

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje?

```
int x, y, Q, p, A, b, c, D, d;  
scanf("%d%d%d%d",&x,&y,&p,&d);  
Q = 2*(x + y);  
c = x*y;  
A = 2*(p+b);  
D = p*b;
```

- Potrebno je preimenovati sve promenljive. Imenovanje je nekonzistentno (velika i mala slova) i nedeskriptivno (nije jasan smisao promenljivih).
- Kod treba da bude konzistentno formatiran (u nekim izrazima postoji a u nekim ne postoji blanko izmedju operatora).

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje?

```
int x, y, Q, p, A, b, c, D, d;  
scanf("%d%d%d%d",&x,&y,&p,&d);  
Q = 2*(x + y);  
c = x*y;  
A = 2*(p+b);  
D = p*b;
```

- Potrebno je preimenovati sve promenljive. Imenovanje je nekonzistentno (velika i mala slova) i nedeskriptivno (nije jasan smisao promenljivih).
- Kod treba da bude konzistentno formatiran (u nekim izrazima postoji a u nekim ne postoji blanko izmedju operatora).
- Koristi se neinicijalizovana promenljiva b - verovatno je u pitanju greška.

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje?

```
int x, y, Q, p, A, b, c, D, d;  
scanf("%d%d%d%d",&x,&y,&p,&d);  
Q = 2*(x + y);  
c = x*y;  
A = 2*(p+b);  
D = p*b;
```

- Potrebno je preimenovati sve promenljive. Imenovanje je nekonzistentno (velika i mala slova) i nedeskriptivno (nije jasan smisao promenljivih).
- Kod treba da bude konzistentno formatiran (u nekim izrazima postoji a u nekim ne postoji blanko izmedju operatora).
- Koristi se neinicijalizovana promenljiva `b` - verovatno je u pitanju greška.
- Izračunavanje je ponovljeno, liči na obim i površinu pravougaonika — bilo bi dobro izračunavanje izdvojiti u odgovarajuće funkcije.

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje

```
if(a>b) if(b>c) return a;  
else if(a>c) return a; else return c;  
else if(b>c) return b; else return c;
```

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje

```
if(a>b) if(b>c) return a;  
else if(a>c) return a; else return c;  
else if(b>c) return b; else return c;
```

- Formatiranje je jako nepregledno, potrebno je propustiti kod kroz odgovarajući alat za formatiranje

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje

```
if(a>b) if(b>c) return a;  
else if(a>c) return a; else return c;  
else if(b>c) return b; else return c;
```

- Formatiranje je jako nepregledno, potrebno je propustiti kod kroz odgovarajući alat za formatiranje
- Kako je u pitanju kod koji računa maksimum tri broja, bolje koristiti funkciju `max`, npr `max(a,max(b,c))` ili standardni algoritam za računanje maksimuma

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje

```
int strcmp(char* s1, char* s2)
{
    while(*s1 && (*s1 == *s2))
    {
        s1++;
        s2++;
    }
    return *s1 - *s2;
}
```

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje

```
int strcmp(char* s1, char* s2)
{
    while(*s1 && (*s1 == *s2))
    {
        s1++;
        s2++;
    }
    return *s1 - *s2;
}
```

- Sama implementacija bi mogla da se poboljša na razne načine (const u argumentima, provera pre prvog dereferenciranja kako ne bi došlo do dereferenciranja null pokazivača, kastovanje da rezultat funkcije bude tipa int)

Da li je moguće bolje dizajnirano ili kvalitetnije implementirano rešenje

```
int strcmp(char* s1, char* s2)
{
    while(*s1 && (*s1 == *s2))
    {
        s1++;
        s2++;
    }
    return *s1 - *s2;
}
```

- Sama implementacija bi mogla da se poboljša na razne načine (const u argumentima, provera pre prvog dereferenciranja kako ne bi došlo do dereferenciranja null pokazivača, kastovanje da rezultat funkcije bude tipa int)
- Ipak, najbolje bi bilo ne koristiti tu funkciju već koristiti bibliotečku funkciju strcmp

Važnost pregleda

Case study

One of our customers set out to test exactly how much money the company would have saved had they used peer review in a certain three-month, 10,000-line project with 10 developers. They tracked how many bugs were found by QA and customers in the subsequent six months. Then they went back and had another group of developers peer-review the code in question. Using metrics from previous releases of this project they knew the average cost of fixing a defect at each phase of development, so they were able to measure directly how much money they would have saved.

The result: Code review would have saved half the cost of fixing the bugs. Plus they would have found 162 additional bugs.

Vrste pregleda koda

U skladu sa nivoom formalnosti ...

... pregledi mogu da budu više ili manje formalni

Formalni pregledi

Formalni pregledi (engl. *formal inspections*)

- Formalni pregledi obuhvataju grupne sastanke (3-6 osoba) na kojima se diskutuje o kodu i rade pregledi (često odštampanog koda).
- To je dosta skupo i vremenski zahtevno, sve manje se koristi jer iako se na ovaj način može pronaći najveći broj defekata u kodu, ovo zahteva previše vremena i angažovanja, a većina firmi to ne može da priušti.

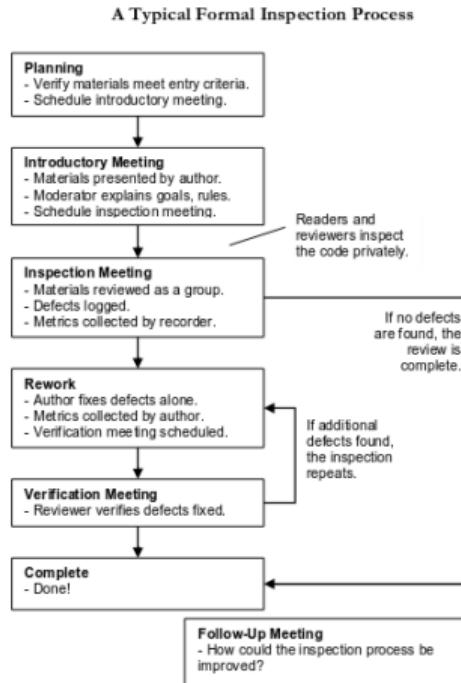


Figure 1: Typical workflow for a "formal" inspection.
Not shown are the artifacts created by the review: The defect log, meeting notes, and metrics log. Some inspections also have a closing questionnaire used in the follow-up meeting.

Neformalni pregledi koda

Podela neformalnih pregleda koda

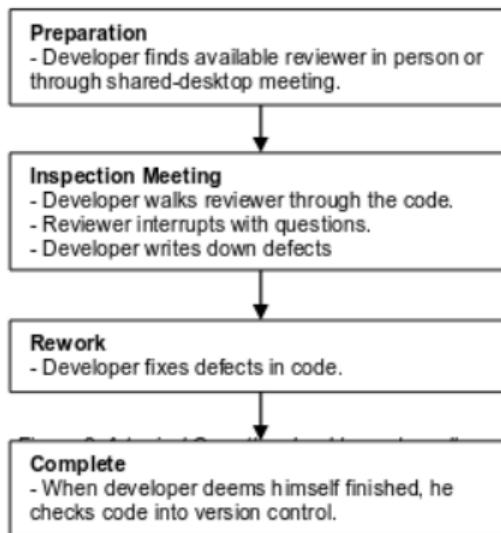
- Neformalni pregledi mogu da budu više i manje neformalni
- Osnovne vrste neformalnih pregleda:
 - pregled preko ramena (engl. *over-the-shoulder reviews*),
 - pregled preko mejla (engl. *e-mail pass-around*),
 - pregled preko alata za pregled koda (engl. *tool-assisted reviews*),
 - programiranje u paru (engl. *pair-programming*).

Neformalni pregledi

Pregled preko ramena

- Najčešći i najneformalniji vid pregleda: programer objašnjava pregledaču šta je u kodu i zašto
- Može da se obavi i preko interneta i deljenog desktop-a, tj ne mora uživo

Over-the-Shoulder Review Process



Neformalni pregledi

Pregled preko ramena

- Najjednostavniji vid, najmanje organizaciono zahtevan, dodatno omogućava i razmenu ideja koje ne bi bile razmenjenjene pisanim putem
- Problemi: nemoguće je ispratiti šta je pregledano, a šta nije, moguće je propustiti neku izmenu ikao je primećeno da treba da se uradi, i iako se konstatuju neki defekti i ako se sprovede akcija da se ti defekti isprave, moguće je da se to uradi pogrešno ili da se uvedu novi defekti (nakon pregleda nema ponovnih provera)

Neformalni pregledi

Pregled preko mejla

- Izmena stigne mejlom pregledačima
- Lakše ukoliko je pregledač na drugoj lokaciji
- Može da bude pre nego što kod uđe u repozitorijum ili automatski da se pošalje nakon što kod uđe u repozitorijum
- I jedna i druga varijanta ima svoje prednosti i mane
- Dugo je ovo bio jedan od najčešćih vrsta pregleda
- Prevaziđen, jer se sada koriste različiti alati koji prevazilaze probleme koji na ovaj način nastaju.

E-Mail Pass-Around Process: Pre Check-In Review

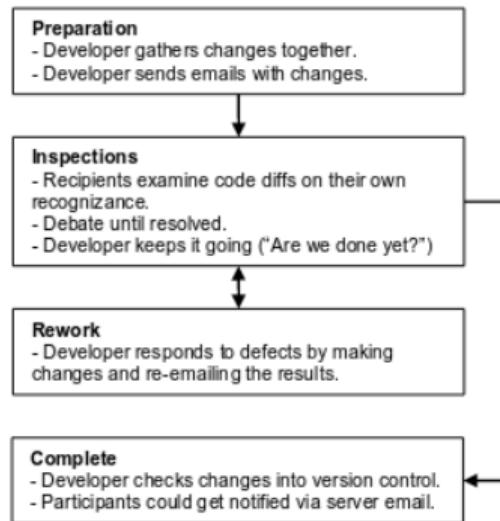


Figure 4: Typical process for an e-mail pass-around review for code already checked into a version control system. These phases are not this distinct in reality because there's no tangible "review" object.

E-Mail Pass-Around Process: Post Check-In Review

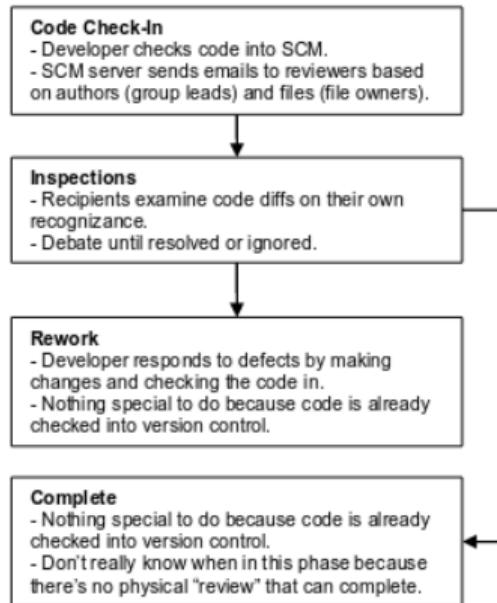


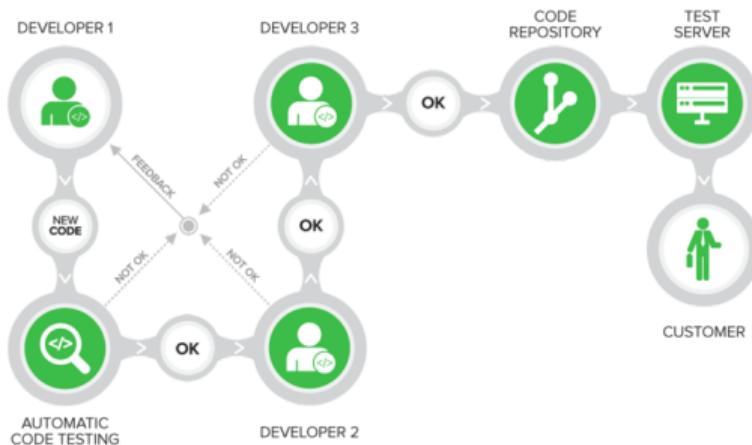
Figure 3: Typical process for an e-mail pass-around review for code already checked into a version control system. These phases are not this distinct in reality because there's no tangible "review" object.

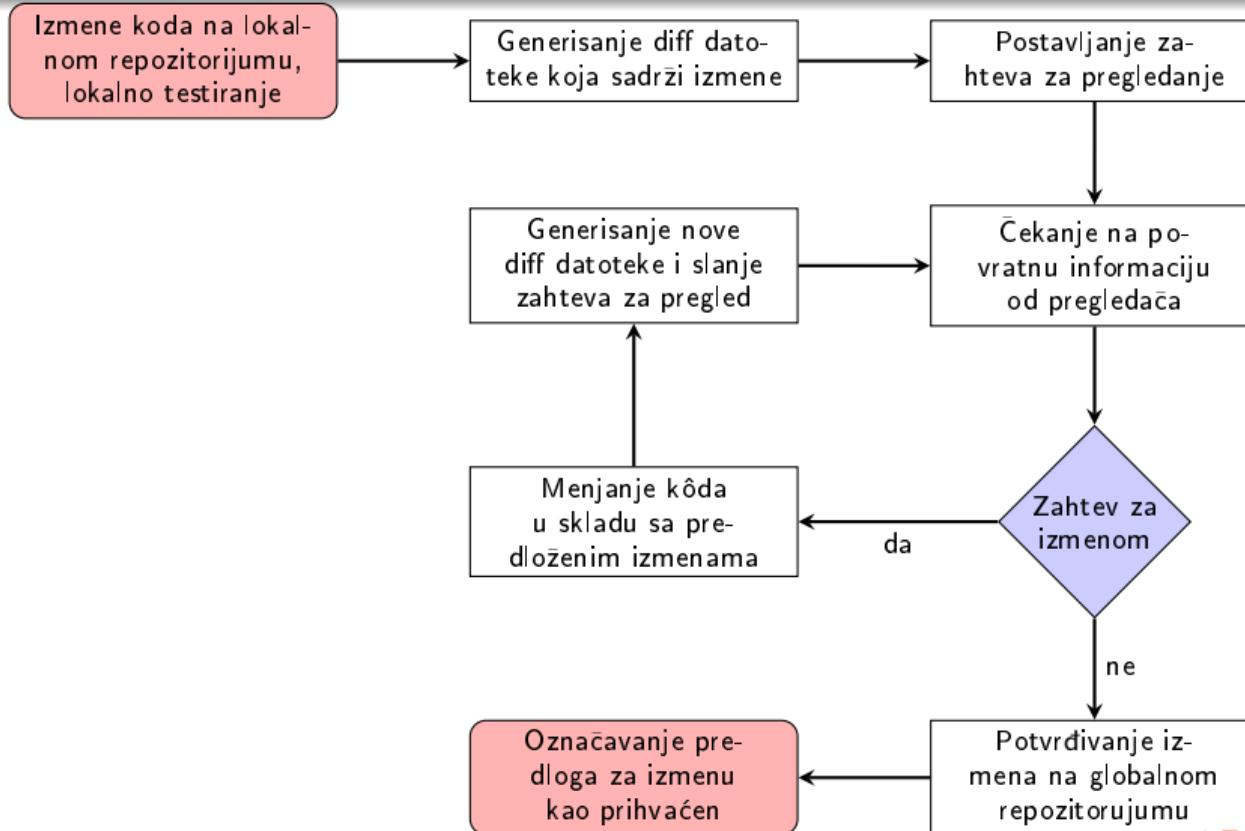
Neformalni pregledi

Pregled preko alata za pregled koda

- Može biti različitog nivoa formalnosti
- Sastavni deo većine agilnih metodologija razvoja softvera
- Pregledi obuhvataju provere koda od strane jednog ili više programera pre nego što kôd uđe u repozitorijum.
- Za preglede su obično zaduženi iskusniji programeri
- Veliki broj alata za podršku: Phabricator, Gerrit, Collaborator, GitLab ...

Proces pregledanja koda





Primer - Phabricator

The screenshot shows a Mozilla Firefox browser window with two tabs open:

- D44304 [MIPS GlobalSel] Select add i32, i32 - Mozilla Firefox
- D44304 [MIPS Glo... - Mozilla Firefox

The main content area displays the Phabricator Differential interface for a patch titled "[MIPS GlobalSel] Select add i32, i32".

Details:

- Authored by PetarAvramovic on Fri, Mar 9, 7:14 AM.
- Needs Review
- Public

Reviewers:

- petarj
- dsanders
- stdardis
- aditya_nandakumar
- qcolombet

SUMMARY:

Add the minimal support necessary to lower a function that returns the sum of two i32 values.
Support argument/return lowering of i32 values through registers only.
Add tablegen for regbankselect and instructionselect.

Diff Detail:

PetarAvramovic created this revision. Fri, Mar 9, 7:14 AM

Actions:

- Edit Revision
- Update Diff
- Download Raw Diff
- Edit Related Revisions...
- Edit Related Objects...
- Subscribe
- Award Token
- Flag for Later

Tags: None

Subscribers: qcolombet, aditya_nandakumar, mgorny and 4 others

Primer - Phabricator

The screenshot shows a Phabricator review interface for a commit titled "D44304 [MIPS GlobalSel] Select add i32, i32". The review has been updated by Petar Avramovic, who created the revision on March 9, 7:14 AM. Herald added subscribers: arichardson, kristof.beijs, rovka, mgorny. Petar Avramovic added a comment on March 16, 8:50 AM, asking "Does anybody have any comment?". dsanders responded on March 19, 10:55 AM, saying "Sorry for taking a while to get to this. It's been a busy couple weeks. The ISel and RegisterBank parts look good to me with a couple convention nits fixed." @aditya_nandakumar and @qcolombet replied with comments. A detailed discussion follows about the naming of DEBUG_TYPE variables and the convention for string matching in MIR. The review was last updated by qcolombet on March 22, 10:18 AM.

D44304 [MIPS GlobalSel] Select add i32, i32 - Mozilla Firefox

Petar Avramovic created this revision. Fri, Mar 9, 7:14 AM

Herald added subscribers: arichardson, kristof.beijs, rovka, mgorny. - View Herald Transcript Fri, Mar 9, 7:14 AM

Petar Avramovic added a comment. Fri, Mar 16, 8:50 AM

Does anybody have any comment?

dsanders added reviewers: aditya_nandakumar, qcolombet. Mon, Mar 19, 10:55 AM

dsanders added subscribers: aditya_nandakumar, qcolombet.

Sorry for taking a while to get to this. It's been a busy couple weeks

The ISel and RegisterBank parts look good to me with a couple convention nits fixed.

@aditya_nandakumar : You're more familiar with GlobalSel's calling convention code than I am. Do you have any comments on that part?

@qcolombet : Do you have any comments on the regbankselect part?

lib/Target/Mips/MipsInstructionSelector.cpp

22 I'd recommend making the 'M' lower case here. Very few DEBUG_TYPE's have uppercase letters and there's no error for picking a value that doesn't exist

lib/Target/Mips/MipsRegisterBanks.td

13 It's not a requirement but by convention the string should match the def (e.g. `def XRegBank :` `RegisterBank<"X", ...>`). ARM doesn't do that because it has a register class by the same name and banks/classes share a namespace in MIR.

qcolombet added a comment. Thu, Mar 22, 10:18 AM

Primer - Phabricator

The screenshot shows a Mozilla Firefox window with two tabs open. The active tab displays a Phabricator code review for a patch titled "D44304 [MIPS GlobalSel] Select add i32, i32". The page lists the revision contents, showing a list of modified files (M) and added files (A M). The modified files include CMakeLists.txt, Mips.td, and several header and source files for MipsCallLowering, MipsInstructionSelector, and MipsRegisterBankInfo. The added files are test cases for instruction-select, irtranslator, legalize, and regbankselect. The interface includes a sidebar with various icons and a top bar with browser controls.

Path	Packages
M lib/Target/Mips/CMakeLists.txt (2 lines)	
M lib/Target/Mips/Mips.td (1 line)	
M lib/Target/Mips/MipsCallLowering.h (36 lines)	
M lib/Target/Mips/MipsCallLowering.cpp (193 lines)	
M lib/Target/Mips/MipsCallLowering.h (5 lines)	
M lib/Target/Mips/MipsCallLowering.cpp (7 lines)	
M lib/Target/Mips/MipsInstructionSelector.cpp (68 lines)	
M lib/Target/Mips/MipsLegalizerInfo.cpp (6 lines)	
M lib/Target/Mips/MipsRegisterBankInfo.h (14 lines)	
M lib/Target/Mips/MipsRegisterBankInfo.cpp (74 lines)	
A M lib/Target/Mips/MipsRegisterBanks.td (13 lines)	
A M test/CodeGen/Mips/GlobalSel/instruction-select/add.mir (35 lines)	
A M test/CodeGen/Mips/GlobalSel/irtranslator/add.ll (16 lines)	
A M test/CodeGen/Mips/GlobalSel/legalizer/add.mir (34 lines)	
A M test/CodeGen/Mips/GlobalSel/llvm-ir/add.ll (12 lines)	
A M test/CodeGen/Mips/GlobalSel/regbankselect/add.mir (34 lines)	

Primer - Phabricator

The screenshot shows a Mozilla Firefox browser window displaying a Phabricator code review page. The URL is https://reviews.llvm.org/D44304. The page title is "D44304 [MIPS GlobalSel] Select add i32, i32 - Mozilla Firefox". The main content is a diff view of the file lib/Target/Mips/MipsInstructionSelector.cpp. The code implements targeting for the Mips instruction selector. A comment from user 'dsanders' suggests changing 'M' to 'm' in the DEBUG_TYPE macro definition. The review status is "Not Done". The browser's toolbar and address bar are visible at the top, and the operating system's taskbar is visible at the bottom.

```
10 // This file implements the targeting of the InstructionSelector class for
11 // Mips.
12 // \todo This should be generated by TableGen.
13 //www.....
14
15 #include "MipsRegisterBankInfo.h"
16 #include "MipsSubtarget.h"
17 #include "MipsTargetMachine.h"
18
19 #include "llvm/Support/Debug.h"
20
21
22 // This file implements the targeting of the InstructionSelector class for
23 // Mips.
24 // \todo This should be generated by TableGen.
25 //www.....
26
27 #include "MipsRegisterBankInfo.h"
28 #include "MipsSubtarget.h"
29 #include "MipsTargetMachine.h"
30
31 #include "llvm/CodeGen/GlobalISel/InstructionSelector.h"
32 #include "llvm/CodeGen/GlobalISel/InstructionSelectorImpl.h"
33 #include "llvm/Support/Debug.h"
34
35 #define DEBUG_TYPE "Mips-isel"
36
37 dsanders Not Done
38
39 I'd recommend making the 'M' lower case here. Very few DEBUG_TYPE's have uppercase
40 letters and there's no error for picking a value that doesn't exist
41
42
43 using namespace llvm;
44
45 namespace {
46
47
48 #define GET_GLOBALISEL_PREDICATE_BITSET
49 #include "MipsGenGlobalSel.inc"
50 #undef GET_GLOBALISEL_PREDICATE_BITSET
51
52 class MipsInstructionSelector : public InstructionSelector {
53 public:
54 }
```

Neformalni pregledi

Programiranje u paru

- Programiranje u paru se može smatrati specijalnom vrstom pregleda
- Programiranje u paru nije uvek prisutno pa se ova vrsta pregleda ne može uvek koristiti
- Programiranje u paru vodi ka kvalitetnijem kodu, ali nekada programeri koji rade zajedno isto razmišljaju i zajedno previđaju greške pa su zato eksterni pregledi ipak neophodni.

Dodatni uticaji pregleda

Uticaj pregleda na programere

- Pravilo (zahtev) da se kôd pregleda pre nego što se ubaci u repozitorijum garantuje da kôd ne može da uđe u repozitorijum nepregledan.
- Znanje da će kôd biti pregledan od strane nekoga u timu, utiče pozitivno i na programere tako da se učini dodatan trud da se sve proveri, dobro dizajnira i da se poštuju pravila kodiranja.
- Pregledi se rade da se provere one stvari koje mašina (automatsko testiranje) ne može da proveri i sprečava loše odluke i loša rešenja da zagađuju osnovnu liniju razvoja (štiti vas od drugih i druge od vas!)

Razmena znanja u oba pravca

Mentorisanje novih programera

- Kada se pridruže novi članovi timu potrebno je da ih neko iskusniji poduči. Pregledi pomažu konverzaciju o kodu. Često timovi imaju svoje skriveno znanje o kodu koje se ispolji tek za vreme pregleda.
- Iako su pregledi prvenstveno od strane iskusnijih programera koji pregledaju posao mlađih programera, pregledi treba da se vrše na svaku stranu. Na primer, novi članovi tima, sa svežim pogledom na stvari i novom perspektivnom, mogu da otkriju neke stare propuste i loše delove koda na koje su se svi navikli i zbog navike ih ne primećuju.

Uloga pregleda u sticanju znanja

Put ka savlađivanju koda projekta

- Pregledi koda pomažu programeru da savlada celokupan kôd projekta, kao i da brže usvoji nove tehnologije i tehnike
- Kako pregledi izlažu programera novim idejama i tehnologijama, na taj način programer uči da stvara sve bolji kôd.

Agilni razvoj softvera

Deljenje znanja

Svi članovi tima imaju koristi od pregleda bez obzira na razvojnu metodologiju. Agilni timovi, dodatno, mogu da imaju dodatne koristi jer njihov posao se na taj način decentralizuje u okviru tima. Suština je da ne postoji jedinstvena ličnost koja zna specifičnosti nekog dela koda, već su svi u sve upućeni. Pregledi omogućavaju i pomažu širenje znanja o kodu u okviru tima.

Zašto je decentralizacija važna?

Niko ne voli da bude jedinstveni kontakt za deo koda (npr, ne može da ode na duži odmor zbog toga). Takođe, niko ne voli da treba da preuzme rad na tuđem kodu, posebno ako je u pitanju nekakva hitna situacija. Pregledi deljenjem znanja kroz tim omogućavaju da svaki član tima može da preuzme i nastavi svaki posao.

11 pravila efikasnog pregledanja

11 pravila efikasnog pregledanja

- Pregledaj manje od 200-400 linija koda od jednom
- Imaj za cilj brzinu pregledanja koja je manja od 300-500 linija po satu
- Planiraj dovoljno vremena za odgovarajuće, sporo pregledanje, ali nikako više od 60 do 90 mintua
- Postaraj se da autori obeleže kod pre nego što pregled počne
- Napravi ciljeve pregledanja koda koji se mogu kvantifikovati i prati metrike kako bi mogao da unaprediš svoj proces pregledanja
- Koristi liste provera koje treba da uradiš, jer se na taj način značajno popravlja rezultat pregleda i za autora i za pregledača

11 pravila efikasnog pregledanja

11 pravila efikasnog pregledanja

- Proveri da su uočeni defekti stvarno i popravljeni
- Neguj dobru kulturu pregleda koda u kojoj se pronalaženje defekata gleda pozitivno
- Budi svesan efekta „Velikog brata“ (Programer može steći utisak da ga neko stalno posmatra, pogotovo ako radi sa alatima za pregledanje. Može da misli da će statistike biti iskorišćene protiv njega i može se fokusirati na poboljšanje statistika umesto na poboljšanje koda)
- Ukoliko ne možeš da postigneš pregled celog koda, pogledaj bar njegov deo (zbog benefita koji donosi „ego“ efekat — svako ulaže dodatni trud kada zna da će njegov kôd neko pregledati)
- Usvoji proces pregleda koda koji koristi alate za pregled koda

Pregled

1 Pregledi koda

2 Automatska statička analiza

3 Literatura

Statička analiza

Automatska statička analiza

Automatska analiza koda bez njegovog izvršavanja sa ciljem pronalaženja defekata u kodu.

Automatska statička analiza

- Simboličko izvršavanje
- Proveravanje modela
- Apstraktna interpretacija
- Analiza vođena kontra-primerima

Pregled

1 Pregledi koda

2 Automatska statička analiza

3 Literatura

Literatura

Linkovi na literaturu

- Metode za pregled koda i njihov značaj
- Best Kept Secrets of Peer Code Review
- Why code reviews matter (and actually save time!)
- 11 proven practices for more effective, efficient peer code review
- Overcoming Pre-Commit Code Review Challenges