VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Pakartotinis kodo panaudojimas pirminio kriptovaliutų platinimo (ICO) išmaniuosiuose kontraktuose

Code review in initial coin offering (ICO) smart contracts

Kursinis darbas

Atliko: 3 kurso 1 grupės studentė
Agnė Mačiukaitė (parašas)

Darbo vadovas: lekt. Gediminas Rimša (parašas)

Vilnius
2018

TURINYS

Įvadas

Programinės įrangos pernaudojimas leidžia naudoti programas keliuose projektuose. Tai yra svarbi strategija programinei įrangai norint padidinti sistemos efektyvumą ir kokybę. Taikant pernaudojamumą programuotojai naudojasi jau įgyvendintu kodu, kurį keičia taip, kad jis atitiktų dabartinio projekto reikalavimus [RR03].

Pirminio kriptovaliuto platinimo (angl. initial coin offering, toliau ICO) metu įmonė parduoda specializuotus kripto-žetonus žadėdami, kad žetonai veiks kaip mainų priemonė gaunat paslaugas įmonės platformoje. Žetonų pardavimas kuria kapitalą pradiniam įmonės platformos kūrimui nors nėra įsipareigojimo dėl būsimos paslaugos kainos (žetonais ar kitaip) [CG18]. Satoshi Nakamoto išleidęs baltąjį popierių (angl. whitepaper) [Nak08] įvykdė ICO ir taip surinko finansavimą pirmąjam blockchain ir kriptovaliutai Bitcoin. Bitcoin - skaitmeniniai pinigai, kurių pavedimai vyksta internete naudojantis decentralizuota vieša duomenų baze - blockchain [Swa15]. Šiuo metu du populiariausi blockchain yra Ethereum ir Bitcoin [LCO+16]. Ethereum be savo kriptovaliutos turi ir kitą svarbų funkcionalumą - išmaniuosius kontraktus - Turing complete programą, kuri leidžia rašyti decentralizuotas aplikacijas [But14]. Solidity - populiariausia kalba naudojama rašyti išmaniesiems kontraktams [Dan17]. Problema - išmaniųjų kontraktų technologijos yra pakankamai jaunos, dėl to pakartotonio kodo panaudojimo bazė dar tik formuojasi. ICO kontraktai yra tiražuojami kopijavimo su modifikacijos būdu.

Programinės įrangos produktų linija (angl. product line software engineering, toliau PLSE) naudojama įmonėse pakartojamumui susijusiuose programinės įrangos produktuose numatyti. PLSE suteikia bendrą architektūrą ir pernaudojamą kodą programinės įrangos kūrėjams [SVB01]. Toks kūrimas susideda iš savybių išskyrimo ir jų įgyvendinimo produkte. Gerai išskirtos produkto ypatybės padeda sukurti lengvai pernaudojamą programą. Savybės turi būti atrinktos atsižvelginat į jų paplitimą bei kintamumą srityje [LKL15]. Naudojantis PLSE produkto kūrėjai gali fokusuotis produkto specifikacijoje, o ne bendrų savybėse [SVB01].

Savybių modeliavimas yra pagrindinis metodas atrinkti bei valdyti bendrąsias ir kintamas savybes produktų linijoje. Programinės įrangos šeimos gyvavimo pradžioje savybių modelis padeda išskirti pagrindines savybes, kurios gelbsti kuriant naują rinką ar norint išlikti jau esamoje. Taip pat savybių modelis leidžia išskirti rizikingas savybes, nuspėti, kokia yra visos programos ar atskirų savybių kaina. Vėliau savybių modeliavimas padeda išskirti variacijos taškus programinės įrangos architektūroje [CHE04]. Savybių modeliavimas yra populiariausias PLSE kūrime nuo pat pirmojo jo pristatymo [KCH+90]. Taip yra todėl, nes savybės yra pakankamai abstraktus konseptas padedantis efektyviai bendrauti suinterasuotoms šalims. Savybių modeliavimas yra intuitivus ir efektyvus būdas žmonėms išreikšti savybių paplitimą ir kintamumą programinės įrangos šeimoje [KL13].

Šio darbo tikslas - ištirti pirminio finansavimo kriptovaliutomis (ICO) išmaniuosius kontraktus, nustatyti, kokios savybės yra pastavios, o kokios - kintamos bei pasiūlyti būdus kodo pernaudojamumui didinti.

Tikslui pasiekti išsikelti uždaviniai:

1. Apžvelgti savybių modeliavimą programinės įrangos produktų linijos sričiai

2. Surinkti virš 100 išmaniųjų kontraktų skirtų ICO
3. Išskirti surinktų kontraktų savybes į pastovias ir kintančias
4. Pasiūlyti ICO išmaniuosius kontraktus pagal išrinktas savybes

1. Savybių modeliavimas

Savybių modeliavime bendri ir kintami bruožai yra modeliuojami iš produkto savybių perspektyvos PLSE, kuri yra suinterestuotų šalių interesas. Originalus savybių modeliavimas - FODA [KCH+90] - paprastas modelis, kuris savybes skirsto pagal tai iš ko jos susideda bei pagal bendrumą ir specializaciją naudojant AND/OR diagramas. Savybės yra suskirstytos į būtinas, alternatyvias ir pasirenkamas pagal bendrus ir kintamus bruožus. Savybių atributai taip pat gali būti dokumentuojami [KL13].

1.1. Savybė

Savybės yra pagrindinis produkto skiriamasis bruožas. Skirtingi srities analizės metodai terminą „savybė" apibūdina šiek tiek kitaip. FODA [KCH+90] savybę apibūdina kaip pastebimą ir skiriamą sistemos charakteristiką, kuri yra matoma įvairioms suinteresuotoms šalims. Svarbu, kad savybių modeliavime turi būti fokusuojamasi ties bendrumo ir skirtumų srityje identifikacija, o ne ties bendrų savybių supratimo išskyrimu. Iš kitos pusės neapibrėžta savybių paskirtis daro sunkumų formuluojant jos semantiką, rezultatų valdymą bei automatinį pagalbo suteikimą.

Skirtumas tarp savybės ir konceptualios abstrakcijos (pvz.: funkcijos, objekto) yra tai, kad funkcijos ir objektai yra naudojami specifikuojant vidines sistemos detales. Kitaip, funkcijos ir objektai yra konceptualios abstrakcijos, kurios yra identifikuojamos iš vidinės sistemos pusės. Savybė - aiškiai matoma pagal charakteristiką, kuri gali išskirti produktą iš kitų. Todėl savybių modeliavimas turi išskirti iš išorės matomas charakteristikas produktuose bendrumo ir kintamumo atžvilgiu, o ne apibūdinti visas produkto modeliavimo detales (pvz.: funkcinis, objektais orientuotas modeliavimas). Suprantant produkto bendrus ir kintamus bruožus galima sukurti pernaudojamas funkcijas ir objektus [LKL15].

1.2. Savybių modelis

Savybėmis orientuotos srities analizės (angl. Feature-Oriented Domain analysis (FODA), toliau FODA) [KCH+90] autoriai apibrėžia savybių modelį, kaip modelį, kuris turi pavaizduoti standartines sistemos šeimos savybes srytyje ir santykius tarp jų. Trumpiau - savybių modelis yra hierarchiškai išskirstytų savybių rinkinys [Bat05]. Struktūrizuoti santykiai tarp savybių, kurie apibūdinami kaip susideda iš (angl. consists of), kurie atvaizduoja logišką savybių suskirtymą, yra interesas. Alternatyvios ar pasirenkamos (angl. optional) savybės turi būti atvaizduotos savybių modelyje. Kiekviena savybė turi būti pavadinta išskirtinai ir jos apibūdinimas turi būti įtrauktas į srities žodyno terminologiją [KCH+90].

Santykiai tarp tėvo ir vaikinės savybės yra kategorizuojami į:

- Ir - visos vaikinės savybės turi būti pasirinktos
- Alternatyva - tik viena vaikinė savybė gali būti pasirinkta
- Ar - viena ar daugiau gali būti pasirinkta

- Būtina - savybė yra privaloma
- Pasirenkama - savybė gali būti pasirenkama

Savybių diagrama yra grafinė savybių modelio reprezentacija. Tai medis, kur primityvios savybės - lapai, pagrininės - mazgai (pav. 1) [Bat05].

FODA [KCH+90] nusako, kad alternatyvios savybės gali būti laikomos kaip specifikavimas bendresnės kategorijos. Bendros savybės atributai yra paveldimi pagal visą jos specifikaciją. Terminas ”alternatyvi savybė” (verčiau ”specifikacijos savybės”) yra naudojama tam, kad apibrėžtų, kad ne daugiau vienos savybės gali būti pasirinkta.

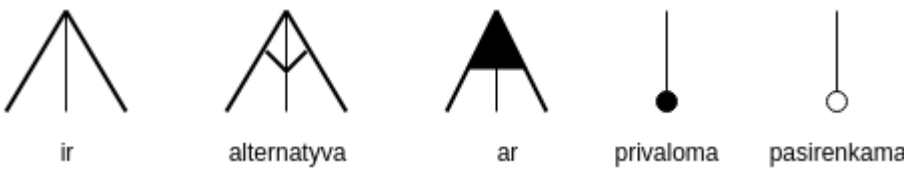
Struktūros taisyklės (pav. 1) apibrėžia semantika tarp savybių, kurios nėra išreikštos diagramoje. Visos pasirenkamos ar alternetyvios savybės, kurios negali būti pasirinktos, kai yra bendra savybė pasirinkta turi būti pažymėtos kaip tarpusavyje nesuderinamos (angl. mutually exclusive with). Visos pasirenkamos ir alternatyvios savybės, kurios turi būti pasirinktos, kai bendra yra pasirinkta, turi būti pažymėtos kaip reikalingos (angl. required).

Pasirenkamų ir alternatyvių savybių pasirinkimas nėra savavališkas. Įprastai jos parenkamos pagal galutinio vartotojo (kliento) tikslus ar interesus. Klausimų rinkinys ir alternatyvus sprendimai renkantis iš pasirenkamų ir alternatyvių savybių yra atvaizduojami naudojantis santykių tipo yra (angl. is-a) forma.

Savybių modelio dokumentacija susideda iš stuktūrinės diagramos hierarchiškai suskaidančios savybes indentifikuojančias pasirenkamas ir alternatyves savybes, savybių apibūdinimo ir taisyklių kompozicijos savybėms.

Savybių modelis tarnauja kaip komunikacija tarp naudotojų ir kūrėjų. Naudotojui savybių modelis teikia informaciją, kokios yra savybės iš kurių gali rinktis ir kada. Kūrėjams savybių modelis identifikuoja poreikius, kuriuos reikia parametrizuotikituose modeliuosi ir programinės įrangos architektūroje ir kaip parametrizacija turi būti atlikta.

Kiti srities modeliai ir programinės įrangos architektūra turi būti apibrėžta aplink standartines savybes. Alternatyvios ir pasirenkams savybės turi būti įtrauktos į modelį ir archtektūrą, bet visada turi būti parametrizuotos su atitinkamomis savybės, kad įsikyšimas į modelį ir architektūra būtų padaromas lengvai.



1 pav. Savybių modelio žymėjimai [Bat05]

1.3. Procesas ir gairės

Savybių analizė susideda iš reikalingų dokumentų surinkimo, savybių išskyrimo, abstrakcijos ir identifikavimo savybių kaip modelį, savybių apibrėžimo, modelio validacijos.

1.3.1. Savybių identifikavimas

Aplikacijos savybes galima išskirti į keturias kategorijas:

- darbo aplinka
- galimybės
- srities technologija
- igyvendimo technika

Metodas fokusuojasi ties savybėmis susijusiomis su aplikacijos galimybėmis. Galimybių savybės dar gali būti suskirstytos į:

- funkcines
- operacines
- pateikimo (prezentacijps)

Funkcinės savybės yra servisai, kurie yra suteikti aplikacijos. Tokios savybės gali būti rastos naudotojo vadove bei reikalavimų dokumentacijoje. Operaicnės savybes - tos kurios susiję su aplikacijos operacijomis (taip pat iš vartotojo perspektyvos); tai yra, kaip naudotojas sčveikauja su aplikacija. Naudotojo vadivas yra geras tokių savybių šaltinis. Prezentacijos - tai kaip ir kokia informacija yra pateikama naudotojui. Tokia informacija randama naudotojo vadove ir reikalavimų specifikacijoje.

Identifikuotos savybės turi būti pavadintos ir konfliktai susiję su vardais turi būti išspręsti. Savybių sinomimai taip pat turi būti įtraukti į srities terminologijos žodyną.

1.3.2. Savybių abstrakcija, klasifikacija, modeliavimas

Sekantis žingsnis identifikavus savybes turėtų būti hierarchinio modelio sukūrimas pagal savybių kklasifikavimą, strūkturizavimą naudojant susideda iš santykį. Ar savybė yra būtina, alternatyvi, pasirenkama turi būti identifikuojama modelyje. Kiekviena savybė modelyje turi būti apibrėžta. Apibūdinimas taip pat turėtų nusakyti ar tai compile-time, activation-time, ar run-time savybė. Tai gali būti nusakyta pagal tai kaip dažnai adaptacija bus daroma.

1.3.3. Savybių modelio validacija

Ar savybių modelis gerai reprezentuoja srities savybes turi būti validuota prieš srities ekspertus ir esančias aplikacijas. Srities ekspertai, kurie konsultavo analizės metu, neturi būti pasirinkti. Taip pat bent viena aplikacija, kuri nebuvo naudota analizėje, turi būti validacijo nustatyti bendrumą ir pritaikomaumą modelio. Jei galima, naujas aplikacijų rinkinys suteiktų geresnę valodaciją, bet galimybė tai padaryti priklausys nuo srities brandos ir finansinių apribojimų.

2. Savybių modeliavimas pirminio kriptovaliutų platinimo išmaniesiems kontraktams

2.1. Savybė

2.2. Savybių modeliavimas

Rezultatai

Išvados

Išvadose ir pasiūlymuose, nekartojant atskirų dalių apibendrinimų, suformuluojamos svarbiausios darbo išvados, rekomendacijos bei pasiūlymai.

Literatūra

[Bat05] Don Batory. Feature Models, Grammars, and Propositional Formulas. *LNCS*, tom. 3714, p. 7–20, 2005. URL: http://www.cs.utexas.edu/ftp/predator/splc05.pdf.

[But14] Vitalik Buterin. A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM. 2014. URL: https://www.weusecoins.com/assets/pdf/library/Ethereum%7B%5C_%7Dwhite%7B%5C_%7Dpaper-a%7B%5C_%7Dnext%7B%5C_%7Dgeneration%7B%5C_%7Dsmart%7B%5C_%7Dcontract%7B%5C_%7Dand%7B%5C_%7Ddecentralized%7B%5C_%7Dapplication%7B%5C_%7Dplatform-vitalik-buterin.pdf.

[CG18] Christian Catalini ir Joshua S Gans. Initial Coin Offerings and the Value of Crypto Tokens, 2018. URL: http://creativecommons.org/licenses/by-nc/4.0/%20https://ssrn.com/abstract=3137213.

[CHE04] Krzysztof Czarnecki, Simon Helsen ir Ulrich Eisenecker. Staged Configuration Using Feature Models. 2004. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.1586%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[Dan17] Chris Dannen. Introducing Ethereum and Solidity Foundations of Cryptocurrency and Blockchain Programming for Beginners Introducing Ethereum and Solidity Foundations of Cryptocurrency and Blockchain Programming for Beginners Introducing Ethereum and Solidity: Foundation. *Library of Congress Control Number*, 2017. DOI: 10.1007/978-1-4842-2535-6. URL: http://smartcontracts.engineer/wp-content/uploads/2017/09/Etherium.pdf.

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak ir A. Spencer Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, tom. 18 numeris 3-4. 1990. ISBN: 1111111111. DOI: 10.1080/10629360701306050.

[KL13] Kyo C. Kang ir Hyesun Lee. Variability Modeling. *Systems and Software Variability Management*, p. 25–42. 2013. ISBN: 978-3-642-36582-9. DOI: 10.1007/978-3-642-36583-6. URL: http://link.springer.com/10.1007/978-3-642-36583-6.

[LCO+16] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena ir Aquinas Hobor. Making Smart Contracts Smarter, 2016. DOI: 10.1145/2976749.2978309. URL: https://www.comp.nus.edu.sg/%7E~%7Dloiluu/papers/oyente.pdf.

[LKL15] Kwanwoo Lee, Kyo C. Kang ir Jaejoon Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. 2015. DOI: 10.1007/3-540-46020-9_5. URL: https://www.researchgate.net/profile/Kwanwoo%7B%5C_%7DLee/publication/221553200%7B%5C_%7DConcepts%7B%5C_%7Dand%7B%5C_%7DGuidelines%7B%5C_%7Dof%7B%5C_%7DFeature%7B%5C_%7DModeling%7B%5C_%7Dfor%7B%5C_%7DProduct%7B%5C_%7DLine%7B%5C_%7DSoftware%7B%5C_%7DEngineering/links/558bdbde08ae591c19d8d3ce.pdf.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[RR03] T. Ravichandran ir Marcus A. Rothenberger. SOFTWARE REUSE STRATEGIES AND COMPONENT MARKETS. *Communications of the ACM*, 46(8), 2003. DOI: 10.1145/859670.859678. URL: https://www.researchgate.net/profile/T%7B%5C_%7DRavichandran/publication/220423696%7B%5C_%7DSoftware%7B%5C_%7Dreuse%7B%5C_%7Dstrategies%7B%5C_%7Dand%7B%5C_%7Dcomponent%7B%5C_%7Dmarkets/links/54201aa60cf2218008d43cdb.pdf.

[SVB01] Mikael Svahnberg, Jilles Van Gurp ir Jan Bosch. On the Notion of Variability in Software Product Lines, 2001. URL: www:%20http://www.ipd.hk-r.se/[msv%7B%5C%%7D7Cjvg%7B%5C%%7D7Cbosch].

[Swa15] Melanie Swan. *Blockchain. Blueprint for a new economy*. 2015, p. 149. ISBN: 978-1-491-92049-7. URL: http://w2.blockchain-tec.net/blockchain/blockchain-by-melanie-swan.pdf.

[Tel94] Astro Teller. Turing Completeness in the Language of Genetic Programming with Indexed Memory, 1994. URL: http://www.astroteller.net/content/3-work/2-papers/17-turing-completeness-in-the-language-of-genetic-programming-with-indexed-memory/turing.pdf.

Sąvokų apibrėžimai

Turing complete - bet kuri sistema, kuri yra pakankamai galinga atpažinti visus galimus algoritmus [Tel94].

Santrumpos

PLSE - programinės įrangos produktų linija (angl. product line software engineering)
ICO - pirminis kriptovaliutų platinimas (angl. initial coin offering)
FODA - Feature-Oriented Domain Analysis [KCH+90]