Energiatehokas koodi

Koodia Suomesta ry., näkemys

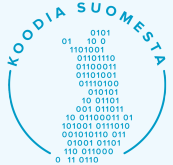
Koodia Suomesta

Haluamme

- Kehittää suomalaista ohjelmointiosaamista varmistaaksemme edellytykset hyvinvointiyhteiskunnallemme
- Kannustaa toimijoita yhteistyöhön ja tarjota verkottumismahdollisuuksia
- Edistää tasa-arvoa ja hyödyntää paremmin osaamispotentiaalia rohkaisemalla aliedustettuja ryhmiä hakeutumaan IT-alalle
- Kehittää suomalaisen ohjelmistoliiketoiminnan kilpailukykyä ja käyttöä niin kotimaassa kuin ulkomailla

Yhdistys on perustettu 2015. Jäseninä on 235 yritystä ja viisi opiskelijajärjestöä.

Koodin energiatehokkuus

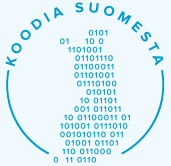


Uskomme koodin energiatehokkuuden olevan tulevaisuudessa merkittävämpi asia kuin se on.

Suomalaisilla vanhemman polven (~40-50v) ohjelmoijilla on osalla on demoscene-kokemusta. Siellä olennaista oli saada hitaalla raudalla mahdollisimman näyttäviä asioita, jolloin tehokkuus oli itseisarvo.

Nykyinen ohjelmistojen tekotapa keskittyy suurelta osin koodarin tehokkuuteen, eli mahdollisimman paljon ominaisuuksia mahdollisimman pienellä hinnalla. Ja tämä johtaa hyvin nopeasti tehottomaan koodiin.

Modernin ohjelmiston rakentaminen



Ohjelmistojen rakentaminen on muuttunut merkittävästi ohjelmistoalan kasvaessa ja kehittyessä.

Ennen ohjelmoijat ymmärsivät koko järjestelmän toiminnan ja pystyivät ohjaamaan sen toimintalogiikka.

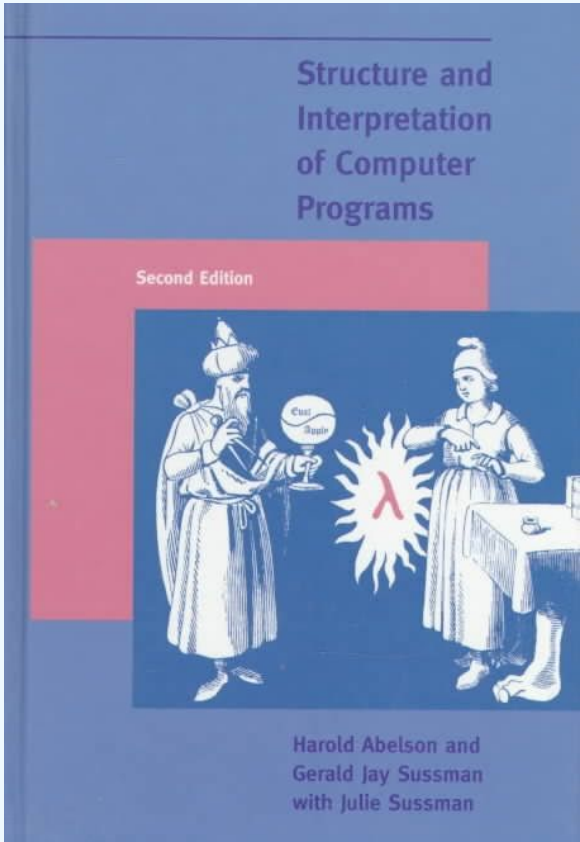
Nykyään ohjelmistot koostetaan suurelta osin valmiista kirjastoista, joiden sisään ei ole aikaa katsoa.

Modernin ohjelmiston rakentaminen

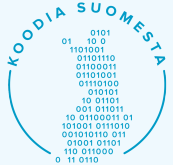
[...] And more importantly, (2) that they felt that the SICP curriculum no longer prepared engineers for what engineering is like today. Sussman said that in the 80s and 90s, engineers built complex systems by combining simple and well-understood parts. The goal of SICP was to provide the abstraction language for reasoning about such systems.

Today, this is no longer the case. Sussman pointed out that engineers now routinely write code for complicated hardware that they don't fully understand (and often can't understand because of trade secrecy.) The same is true at the software level, since programming environments consist of gigantic libraries with enormous functionality. [...]

http://lambda-the-ultimate.org/node/5335



Tehokkuus vs. käytettävyys

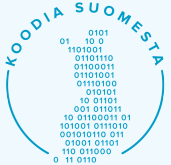


Hyvin harva ohjelmoija miettii käyttämiensä kirjastojen tehokkuutta, ellei lopputuloksessa ole tehokkuusongelmaa – hitautta, pätkimistä, muistin loppumista, yms.

Sen sijaan fokuksessa on saada mahdollisimman nopeasti ja/tai mukavasti ratkaisu aikaan. Pääpaino on ohjelmoijan viihtyvyydessä, ei lopputuloksen laadussa.

Tähän kannustaa erityisesti Suomessa konsulttiyritysten käyttämä laskutusmalli. Työt tehdään tunti tunnista ja yrityksellä ei ole mitään takaisinkytkentää tai vastuuta järjestelmän tehokkuudesta. Mikäli asiakkaalle syntyy suuret käyttökustannukset, niin se on ainoastaan asiakkaan ongelma.

Kirjastojen haaste



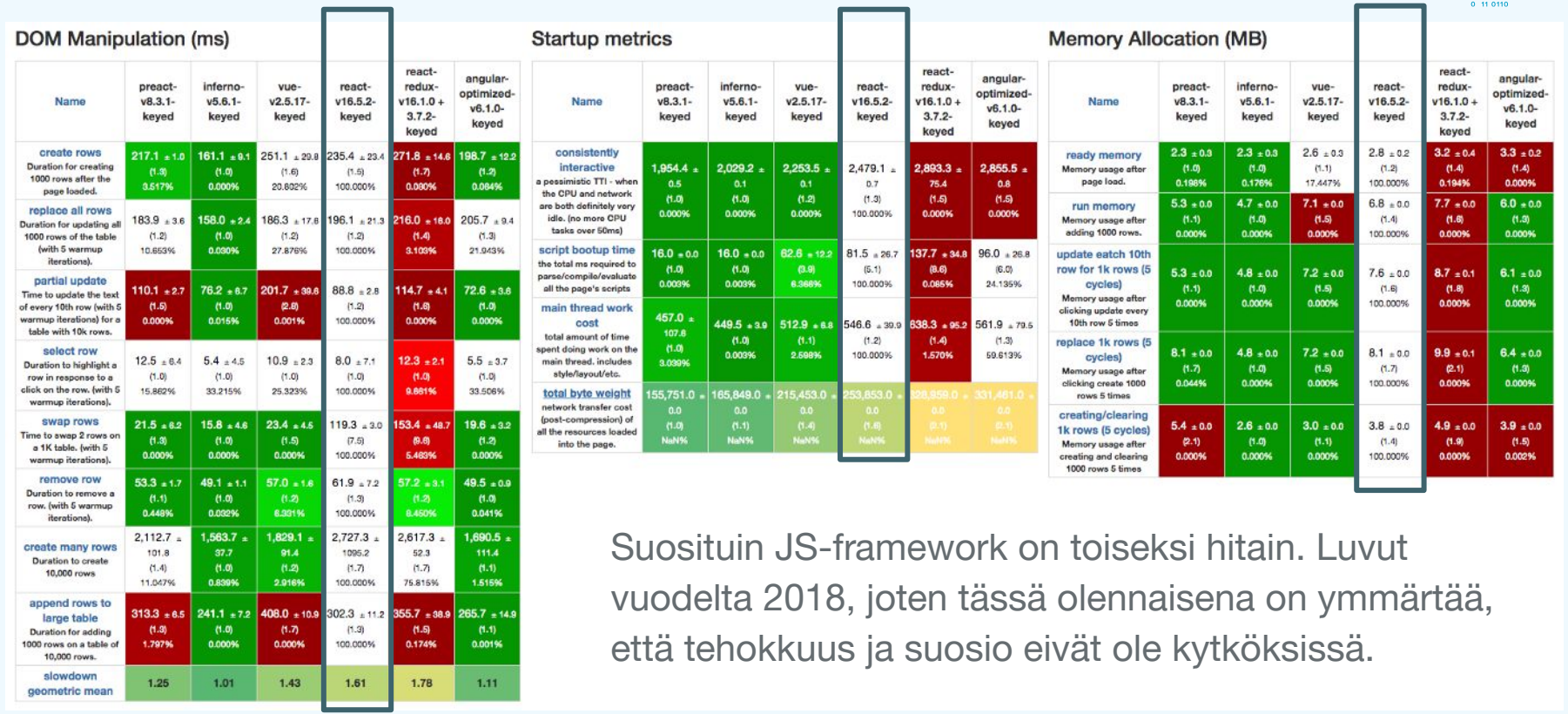
Koska ohjelmat kootaan kirjastoista ja niiden käyttäminen johtaa tehokkaaseen ratkaisuun ohjelmoijan kannalta, kirjastojen merkitys ohjelmistojen tehokkuudessa on valtava.

Kirjastot suunnitellaan yleiseen käyttöön, jolloin niissä on paljon ehtoja, tuloarvojen testausta ja yleisyyttä – mikä johtaa aina huonompaan tehokkuuteen.

Osa kirjastoista on kirjoitettu todella huonosti ja kirjastoja on paljon sisäkkäin.

Keskiverto-ohjelmoija ei yleensä selvitä tai voi tietää, onko valittu kirjasto energiatehokas.

Esimerkki



DOM Manipulation (ms)

Name	preact-v8.3.1-keyed	inferno-v5.6.1-keyed	vue-v2.5.17-keyed	react-v16.5.2-keyed	react-redux-v16.1.0 + 3.7.2-keyed	angular-optimized-v6.1.0-keyed
create rows Duration for creating 1000 rows after the page loaded.	217.1 ±1.0 (1.3) 3.517%	161.1 ±9.1 (1.0) 0.000%	251.1 ±29.8 (1.6) 20.802%	235.4 ±23.4 (1.5) 100.000%	271.8 ±14.6 (1.7) 0.000%	198.7 ±12.2 (1.2) 0.084%
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	183.9 ±3.6 (1.2) 10.653%	158.0 ±2.4 (1.0) 0.000%	186.3 ±17.6 (1.2) 27.876%	196.1 ±21.3 (1.2) 100.000%	216.0 ±16.0 (1.4) 3.109%	205.7 ±9.4 (1.3) 21.943%
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	110.1 ±2.7 (1.5) 0.000%	76.2 ±6.7 (1.0) 0.015%	201.7 ±39.6 (2.6) 0.001%	88.8 ±2.8 (1.2) 100.000%	114.7 ±4.1 (1.6) 0.000%	72.6 ±3.6 (1.0) 0.000%
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	12.5 ±6.4 (1.0) 15.862%	5.4 ±4.5 (1.0) 33.215%	10.9 ±2.3 (1.0) 25.323%	8.0 ±7.1 (1.0) 100.000%	12.3 ±2.1 (1.0) 9.881%	5.5 ±3.7 (1.0) 33.506%
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	21.5 ±8.2 (1.3) 0.000%	15.8 ±4.6 (1.0) 0.000%	23.4 ±4.5 (1.5) 0.000%	119.3 ±3.0 (7.5) 100.000%	153.4 ±48.7 (9.6) 5.489%	19.6 ±3.2 (1.2) 0.000%
remove row Duration to remove a row. (with 5 warmup iterations).	53.3 ±1.7 (1.1) 0.449%	49.1 ±1.1 (1.0) 0.032%	57.0 ±1.6 (1.2) 6.331%	61.9 ±7.2 (1.3) 100.000%	57.2 ±3.1 (1.2) 0.450%	49.5 ±0.9 (1.0) 0.041%
create many rows Duration to create 10,000 rows	2,112.7 ±101.8 (1.4) 11.047%	1,563.7 ±37.7 (1.0) 0.839%	1,829.1 ±91.4 (1.2) 2.916%	2,727.3 ±1095.2 (1.7) 100.000%	2,617.3 ±52.3 (1.7) 75.815%	1,690.5 ±111.4 (1.1) 1.515%
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	313.3 ±6.5 (1.3) 1.797%	241.1 ±7.2 (1.0) 0.000%	408.0 ±10.9 (1.7) 0.000%	302.3 ±11.2 (1.3) 100.000%	355.7 ±38.9 (1.5) 0.174%	265.7 ±14.9 (1.1) 0.001%
slowdown geometric mean	1.25	1.01	1.43	1.61	1.78	1.11

Startup metrics

Name	preact-v8.3.1-keyed	inferno-v5.6.1-keyed	vue-v2.5.17-keyed	react-v16.5.2-keyed	react-redux-v16.1.0 + 3.7.2-keyed	angular-optimized-v6.1.0-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	1,954.4 ± 0.5 (1.0) 0.000%	2,029.2 ± 0.1 (1.0) 0.000%	2,253.5 ± 0.1 (1.2) 0.000%	2,479.1 ± 0.7 (1.3) 100.000%	2,893.3 ± 75.4 (1.5) 0.000%	2,855.5 ± 0.8 (1.5) 0.000%
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	16.0 ± 0.0 (1.0) 0.003%	16.0 ± 0.0 (1.0) 0.003%	62.6 ± 12.2 (3.9) 6.368%	81.5 ± 26.7 (5.1) 100.000%	137.7 ± 34.8 (8.6) 0.065%	96.0 ± 26.8 (6.0) 24.135%
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	457.0 ± 107.6 (1.0) 3.039%	449.5 ± 3.9 (1.0) 0.003%	512.9 ± 6.8 (1.1) 2.598%	546.6 ± 39.9 (1.2) 100.000%	838.3 ± 95.2 (1.4) 1.570%	561.9 ± 79.5 (1.3) 59.613%
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	155,751.0 ± 0.0 (1.0) NaN%	165,849.0 ± 0.0 (1.1) NaN%	215,453.0 ± 0.0 (1.4) NaN%	253,853.0 ± 0.0 (1.6) NaN%	328,959.0 ± 0.0 (2.1) NaN%	331,461.0 ± 0.0 (2.1) NaN%

Memory Allocation (MB)

Name	preact-v8.3.1-keyed	inferno-v5.6.1-keyed	vue-v2.5.17-keyed	react-v16.5.2-keyed	react-redux-v16.1.0 + 3.7.2-keyed	angular-optimized-v6.1.0-keyed
ready memory Memory usage after page load.	2.3 ±0.0 (1.0) 0.198%	2.3 ±0.0 (1.0) 0.176%	2.6 ±0.0 (1.1) 17.447%	2.8 ±0.2 (1.2) 100.000%	3.2 ±0.4 (1.4) 0.194%	3.3 ±0.2 (1.4) 0.000%
run memory Memory usage after adding 1000 rows.	5.3 ±0.0 (1.1) 0.000%	4.7 ±0.0 (1.0) 0.000%	7.1 ±0.0 (1.5) 0.000%	6.8 ±0.0 (1.4) 100.000%	7.7 ±0.0 (1.6) 0.000%	6.0 ±0.0 (1.3) 0.000%
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	5.3 ±0.0 (1.1) 0.000%	4.8 ±0.0 (1.0) 0.000%	7.2 ±0.0 (1.5) 0.000%	7.6 ±0.0 (1.6) 100.000%	8.7 ±0.1 (1.8) 0.000%	6.1 ±0.0 (1.3) 0.000%
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	8.1 ±0.0 (1.7) 0.044%	4.8 ±0.0 (1.0) 0.000%	7.2 ±0.0 (1.5) 0.000%	8.1 ±0.0 (1.7) 100.000%	9.9 ±0.1 (2.1) 0.000%	6.4 ±0.0 (1.3) 0.000%
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	5.4 ±0.0 (2.1) 0.000%	2.6 ±0.0 (1.0) 0.000%	3.0 ±0.0 (1.1) 0.000%	3.8 ±0.0 (1.4) 100.000%	4.9 ±0.0 (1.9) 0.000%	3.9 ±0.0 (1.5) 0.002%

Suosituin JS-framework on toiseksi hitain. Luvut vuodelta 2018, joten tässä olennaisena on ymmärtää, että tehokkuus ja suosio eivät ole kytköksissä.

https://medium.com/@ajmeyghani/javascript-frameworks-performance-comparison-c566d19ab65b

Vaikutukset kansantalouteen

Koska nyt ohjelmointialan fokus on pitkälti ohjelmoijan tuottavuudessa, järjestelmän tehokkuus on toissijaista.

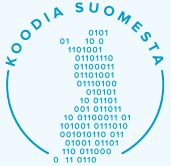
Tämä johtaa suurempiin palvelinkustannuksiin.

Suurin osa palvelininfrasta on yhdysvaltalaisten yritysten pilvipalveluissa.

Suomesta vuodetaan koko ajan rahaa ulos infran käytön takia. Tehottomat ohjelmat ja ratkaisut kasvattavat tätä vuotoa.

Ohjelmoijaa ja hänen edustamaansa yritystä ei kiinnosta asiakkaan rahankäyttö, ellei asiakas nosta asiaa itse esiin. Ja asiakas ei välttämättä tiedä paremmasta.

Ratkaisuja



Koodin energiatehokkuus tulisi nostaa – tavalla tai toisella – kriteeristöksi valita toimittajia.

Julkisella vallalla on tähän Suomessa erinomaiset mahdollisuudet, koska se on merkittävä ostaja konsulttiyrityksiltä ja muilta suomalaisilta ohjelmistoalan toimijoilta.

Esimerkiksi JHS-suositus energiatehokkaasta koodista, tämä vaatimuksena tekijöille ja sen todentaminen vuoden päästä ohjelmiston valmistumisesta ulkopuolisen konsultin toimesta – ja tähän liitetty keppi tai porkkana.

- Malli vertautuu tietoturvaselvityksiin

Ratkaisuja

Merkittävässä käytössä olevien kirjastojen tehokkuusoptimointi tuottaisi energiasäästöä kaikissa niitä käyttävissä sovelluksissa

- Siirtymä vaatii toki sovelluksen päivittämisen

Olisiko Suomen julkinen sektori halukas laittamaan palkkioita tiettyjen kirjastojen optimointiin?

- Olisiko business case järkevä – palkinto vs. saavutettava säästö pilvipalvelukuluissa?
- Voisiko asiaa lähestyä rahaston kautta esimerkiksi EU-tasolla

Ratkaisuja

Energiatehokkuustietoisuuden nostaminen ohjelmoijien keskuudessa.

Suomesta löytyy muutamia hyviä esimerkkejä tehokkaista ratkaisuista, kuten RE:DOM tai Aviamaps.

Kuinka näitä voitaisiin nostaa esimerkeiksi, joista muut voisivat ottaa mallia?

Miten ohjelmoijien koulutuksessa voidaan ottaa energiatehokkuus huomioon?

Kiitos.

Kysymyksiä? Kommentteja?