HÖGSKOLAN
I GÄVLE

AKADEMIN FÖR TEKNIK OCH MILJÖ
Avdelningen för datavetenskap och samhällsbyggnad

Grön mjukvaruutveckling

Förhållandet mellan gröna val och standarder

Felicia Söder

2023

Examensarbete, Grundnivå (yrkesexamen), 15 hp
Datavetenskap
Dataingenjörsprogrammet

Handledare: Niklas Humble
Examinator: Carina Pettersson

Förord

Jag vill ge ett stort tack till min handledare Fredrik Tåneland och alla på Monitor ERP System för hjälpen och möjligheten att genomföra mitt examensarbete på företaget.

Jag vill även tacka min handledare Niklas Humble för kontinuerlig vägledning i skrivandet och alltid snabba svar.

Till sist vill jag tacka min fru Sara Söder som alltid är mitt största stöd och sett till att jag haft allt jag behöver genom hela arbetsprocessen

Sammanfattning

Att minska energikonsumtionen inom mjukvaruutveckling är ett högst aktuellt ämne då datacenter och dataöverföringar står för 0,9% av de globala växthusgasutsläppen. Syftet med den här studien var att undersöka förhållandet mellan gröna val och standarder i system med många utvecklare. Detta har gjorts genom att undersöka hur utvecklare på Monitor ERP System AB ser på hållbarhet under utvecklingsprocessen, samt hur en algoritms tidskomplexitet och val av primitiva datatyper och datastrukturer påverkar en mjukvarulösnings energikonsumtion.

Arbetet genomfördes med mixad metod och använde intervjuer och experiment som datainsamlingsmetoder. Intervjuer hölls med utvecklare på Monitor ERP System för att undersöka vad som påverkade utvecklingsprocessen och skapa en uppfattning om var gröna val kunde ta plats. Experiment genomfördes för att undersöka hur algoritmers tidskomplexitet och val av primitiva datatyper och datastrukturer påverkar energikonsumtionen i en mjukvarulösning.

Resultatet från intervjuerna visade att hållbarhet inte togs in som en faktor i utvecklingsprocessen på grund av kunskapsbrist och att prestanda var en högt prioriterad faktor. Detta kombinerades med experimentets resultat att energikonsumtionen av en mjukvarulösning berodde på programinstruktioners exekveringstid i kombination med en algoritms tidskomplexitet. Slutsatsen av arbetet är att grön mjukvaruutveckling kan tas in i system med standarder genom att skapa nya gröna standarder med fokus på prestanda och lättillgänglig sammanställning av gröna val.

Nyckelord: Grön mjukvaruutveckling, Kodstandardisering, Tidskomplexitet, Primitiva datatyper, Datastrukturer.

Abstract

Reducing the energy consumption in software development is a highly relevant subject, as data centers and data transmissions make up 0.9% of the global carbon emissions. The purpose of this study was to examine the relationship between green choices and standards in systems with many developers. This was investigated by examining how developers at Monitor ERP System AB view sustainability during the development process, as well as how an algorithm's time complexity and choices of primitive data types and data structures influence a software solution's energy consumption.

The study was conducted using mixed method with interviews and an experiment as data collection methods. Interviews were held with developers at Monitor ERP System to determine what affected the development process and create a perception for where green choices could take place. Experiments were conducted to investigate how an algorithm's time complexity and choices of primitive data types and data structures affect the energy consumption of a software solution.

The interview results show that sustainability was not considered during the development process due to a lack of knowledge and that performance was a heavily prioritized factor. This was combined with the experiment results that energy consumption was dependent on the execution time of program instructions in combination with an algorithm's time complexity. The conclusion was that green software development can take place in systems with standards by creating new green standards, focusing on performance and easy access to a collection of green choices.

Key words: Green software development, Coding standards, Time complexity, Primitive data types, Data structures.

Innehållsförteckning

1 Introduktion

I en alltmer digitaliserad värld så ökar behovet av energi. Enligt internationella energibyrån (IEA) står datacenter och dataöverföringar vardera för 1–1,5% av den globala energikonsumtionen, med ett bidrag till de globala energirelaterade växthusgasutsläppen på 0,9% [1]. Detta kan jämföras med flygindustrin som bidrar med 2% av dessa utsläpp [2].

Fokus har länge legat på att göra hårdvara energieffektiv [3] och att använda sig av förnybar energi vid drift av till exempel datacenter [1]. I vissa fall återanvänder man sig även av energi och värme från datacenter för att värma upp bostäder i närliggande områden för att minska energisvinnet vid nedkylning av ett datacenters servrar [1]. Idéer om hur även mjukvara kan påverka energikonsumtionen och effektivisera användandet för en positiv påverkan på växthusgasutsläppen växer fram [3].

Grön mjukvaruforskning genomförs med två syften [4]. Antingen så syftar forskningen till att koden i sig ska vara grön och ha ett lågt koldioxidavtryck, eller att mjukvarans syfte direkt minskar en applikations miljöpåverkan. I denna studie undersöks det förstnämnda, hur koden i sig kan göras mer energieffektiv och hur detta kan göras i system med många utvecklare.

Denna studie ämnar undersöka möjligheten att kunna göra gröna val under utvecklingsprocessen i ett större system med många utvecklare, samt hur en algoritms tidskomplexitet, primitiva datatyper och datastrukturer (dessa beskrivs närmare under rubrik 2 Bakgrund) påverkar energikonsumtionen av en mjukvarulösning. Ett antagande görs i denna studie: för att påverka ett systems energikonsumtion krävs individuella, gröna val.

För att ta reda på var grön mjukvaruutveckling kan appliceras på stora mjukvarusystem med många utvecklare så genomförs intervjuer med utvecklare på Monitor ERP System AB, i framtiden refererat till som Monitor, samt experiment för hur energikonsumtionen påverkas av olika val. Monitor är ett produktbolag som utvecklar och säljer ett affärssystem ”som täcker alla aktiviteter i ett tillverkande företag”, som idag är installerat hos över 5 000 företag [5]. Utvecklingsdelen av organisationen består av två utvecklingsavdelningar och totalt 12 team med många utvecklare.

1.1 Syfte och frågeställningar

Syftet med detta examensarbete är att undersöka förhållandet mellan gröna val och standarder i utvecklingen av system med många utvecklare. Detta genomförs för att ta reda på hur grön mjukvaruutveckling kan tas in i system med standarder. Till stöd för arbetet så har följande frågeställning använts:

- Hur ser utvecklare på Monitor på hållbarhet i förhållande till utvecklingsprocessen av system?
- Hur påverkas en mjukvarulösnings energikonsumtion av algoritmers tidskomplexitet och val av primitiva datatyper och datastrukturer?

2 Bakgrund

Grön mjukvaruutveckling handlar om att minska energikonsumtionen för mjukvarulösningar [6] och är ett växande koncept [7]. Tidigare forskning har gjorts på flertalet aspekter. I denna studie har den tidigare forskningen avgränsats till två delar: 1) Hur mäter man energikonsumtion för mjukvarulösningar? 2) Hur påverkar algoritmer, tidskomplexitet, primitiva datatyper, datastrukturer och designmönster en mjukvarulösnings energikonsumtion?

2.1 Energimätning av mjukvara

Flertalet energimätningsverktyg har använts och tagits fram under studier inom grön mjukvaruutveckling. Enligt Georgiou et al. [8] så används två metoder för mätningarna: 1) Indirekt mätning med estimeringsmodeller och prestandaräknare,
2) Direkt mätning med hårdvarusensorer. Running Average Power Limit (RAPL), som används i denna studie använder sig av den första metoden [8].

RAPL är ett gränssnitt som inkluderas med de flesta moderna Intel processorer [9]. RAPL används för att rapportera energikonsumtionen för ett system över viss tid och med olika verktyg kan man utvinna datat på olika sätt. I ett Linux- eller Mac-system så kan man använda sig av RAPL på kommandoraden som en del av en installation av Mozilla Firefox [10]. Resultatet av en körning med RAPL på kommandoraden visas i figur 1 med antalet poster beroende på antalet samplingar man bestämt, i detta fall fem, och hur ofta en sampling ska tas, i detta fall varje sekund. I figur 1 visas att energimätningen delas upp i kärnor, GPU och RAM med en övrig post för resterande energikonsumtion.

```
    total W = _pkg_ (cores + _gpu_ + other) + _ram_ W
#01  1.02 W =  0.75 ( 0.24 +  0.01 +  0.51) +  0.26 W
#02  0.88 W =  0.63 ( 0.14 +  0.01 +  0.47) +  0.25 W
#03  1.08 W =  0.81 ( 0.24 +  0.01 +  0.56) +  0.27 W
#04  0.83 W =  0.58 ( 0.12 +  0.01 +  0.44) +  0.25 W
#05  0.89 W =  0.63 ( 0.16 +  0.01 +  0.46) +  0.26 W

5 samples taken over a period of 5.000 seconds

Distribution of 'total' values:
           mean =  0.94 W
        std dev =  0.09 W
  0th percentile =  0.83 W (min)
  5th percentile =  0.83 W
 25th percentile =  0.88 W
 50th percentile =  0.89 W
 75th percentile =  1.02 W
 95th percentile =  1.08 W
100th percentile =  1.08 W (max)
```

Figur 1: Resultatet av en mätning med RAPL i fem sekunder

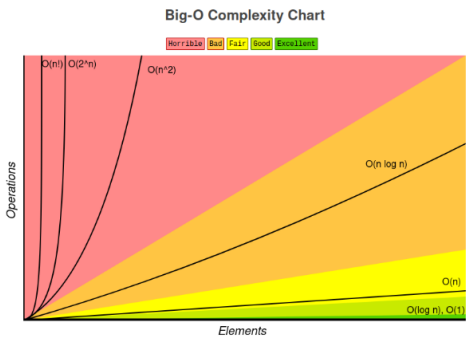
För att upprätthålla validitet (validitet beskrivs närmare under rubrik 3.3.1 Validitet) vid energimätning sammanfattas följande steg: Först rekommenderar Ghaleb [11] att man ska köra ett basprogram N antal gånger under mätning för att sedan upprepa detta efter att man gjort kodförändringar. Skillnaden mellan dessa mätningar visar hur energikonsumtionen påverkades av kodförändringarna. Detta följs upp av Noureddine & Rajan [12] som i sin studie upprepar varje test 100 000 gånger i en loop för att uppmäta tillräckliga data. Utöver repetitioner skriver Jagroep et al. [13] att man innan mätning bör starta om testmiljön, stänga av onödiga applikationer och ge mätdatorn tid att kyla ner hårdvarukomponenter och sluta arbeta innan mätningen genomförs.

2.2 Algoritmer och tidskomplexitet

Algoritmer definieras som en lista av sekventiella instruktioner för hur ett program ska exekveras och återfinns i alla delar av ett system [14]. Till exempel används algoritmer för alla operationer som genomförs på en datastruktur, såsom att hämta och lägga in data. Vanliga exempel på mjukvarualgoritmer är olika sorteringsalgoritmer som på olika sätt och med olika tidskomplexitet sorterar en datamängd baserat på listan av instruktioner [15].

Tidskomplexitet är ett teoretiskt matematiskt mått för hur tiden av en algoritm förändras med dess storlek när den går mot oändligheten [16]. Storleken definieras ofta efter antalet värden den hanterar. Det relevanta vid tidskomplexitet anses inte vara hur lång tid det tar utan hur utvecklingen av tid vid storleksökning ser ut.

När en tidskomplexitet kopplas till en algoritm använder man sig av olika möjliga utfall. Det sämsta utfallet definieras med Big Oh (O), medelutfallet med Big Theta (Θ) och bästa utfallet med Big Omega (Ω) [14]. De tidskomplexiteter som brukar användas inom datavetenskapen är i stigande ordning: konstanta (1), logaritmiska ($log\ n$), linjära (n), linjärlogaritmiska ($n\ log\ n$), kvadratiska (n^2), exponentiella (2^n) och faktoriella ($n!$). Hur tiden utvecklas för dessa tidskomplexiteter visualiseras i figur 2 [17].



Figur 2: Diagram över tidskomplexiteters tidsutveckling vid storleksökning [17]

2.3 Primitiva datatyper

Primitiva datatyper är grundbyggstenar för programmeringsspråk som beskriver vilken typ av data som hanteras av till exempel en variabel [14]. En primitiv datatyp skiljer sig från ett objekt då den inte håller i någon funktionalitet men kan hanteras i aritmetiska operationer [14]. Varje primitiv datatyp har ett maxvärde som sätter storleken för dem baserat på hur många ettor och nollor som krävs för att skriva maxvärdet binärt. Vilka primitiva datatyper som finns i ett språk varierar [18]. I Java används till exempel de primitiva datatyperna i tabell 1 .

Tabell 1: Primitiva datatyper i Java [19]

Datatyp	**Storlek**
byte	8 bitar
short	16 bitar
int	32 bitar
long	64 bitar
float	32 bitar
double	64 bitar
boolean	1 bit
char	16 bitar

2.4 Datastrukturer

Datastrukturer är sätt att strukturera samlingar av data [14]. Detta kan göras på flera sätt och de datastrukturer som undersöks i denna studie är: arrayer, länkade listor, stackar, köer och binära träd implementerade som mappar och set. Dessa beskrivs närmare i tabell 2.

Tabell 2: Beskrivningar för vanliga datastrukturer

Begrepp	**Beskrivning**
Array	Arrayer är definierade som att ha en oföränderlig storlek och lagra alla värden mot ett heltalsindex som används för all hantering av värdet [14].
Länkade listor	Länkade listor består av noder som håller i ett värde och en referens till nästa nod i listan [15].
Stackar	Stackar lagrar värden på samma sätt som länkade listor men utgår från principen först in sist ut vid insättning och uttagning av data [15].
Köer	Köer lagrar värden på samma sätt som länkade listor men utgår från principen först in först ut vid insättning och uttagning av data [15].
Binära träd	Binära träd består av noder med ett värde och två referenser till nästkommande noder och är en hierarkisk datastruktur [15].
Mappar	Mappar kan implementeras som både listor och träd. De lagrar data i tupler av nycklar och värden där nycklarna är unika och lagras i den övergripande datastrukturen [15].
Set	Set kan implementeras som både listor och träd och kan endast innehålla unika värden [15].

2.4.1 Studier inom grön mjukvaruutveckling

I åtminstone tre studier har Javas datastrukturer undersökts efter hur deras operationer konsumerar energi efter storleken på datamängder. Syftet med studierna var att ta reda på hur man väljer rätt datastruktur ur ett energiperspektiv.

Pinto et al. [20] undersöker hur Javas datastrukturer påverkar energikonsumtionen utifrån olika implementationer, operationer, konfigurationer och antal trådar. Resultaten visade att olika operationer för samma datastruktur konsumerade olika mycket energi och att man med små förändringar kunde minska energikonsumtionen.

Pereira et al. [21] undersöker hur olika datastrukturer i Java: Sets, Lists och Maps, påverkar energikonsumtionen vid olika datamängder. De mätte energikonsumtionen för de olika datastrukturerna och sammanställde resultaten för varje operation. De använde sedan fem studentprojekt för att mäta om man vid byte från energiineffektiva datastrukturer till energieffektiva datastrukturer kunde minska energikonsumtionen av projekten. När en datastruktur med dåliga energiresultat hittades i projekten så byttes det ut till ett med bra energiresultat. Resultaten av dessa byten visade en minskning av energikonsumtion på 4,37% – 11,05%.

Hasan et al. [22] undersöker hur mycket olika operationer på olika platser i Javas datastrukturer kostar i energi efter storleken på datamängden. De jämförde även datastrukturer från externa bibliotek. För att ta reda på hur olika datastrukturer påverkar energikonsumtionen i en applikation så bytte de ut utvalda datastrukturer i sex applikationer. Resultaten visade på en energikonsumtionsökning på upp till 300% då fel datastruktur valts och en minskning på upp till 38% när rätt datastruktur valts.

2.5 Kodstandarder

Enligt Martin [23] bör alla team följa en kodstandard. Han beskriver en kodstandard som en samling av riktlinjer som definierar allt från hur man döper klasser, metoder och variabler till var man skriver en hakparentes.

Designmönster är ett väletablerat sätt att standardisera kod. Designmönster definieras enligt Gamma et al. [24] som generella lösningar på återkommande utvecklingsproblem. Dessa används frekvent inom mjukvaruutveckling och sträcker sig utanför programmeringsspråk. De delas in i tre kategorier som baseras på vilken sorts problem designmönstret löser: skapande, strukturella och beteendemönster [24].

2.5.1 Studier inom grön mjukvaruutveckling

Tidigare studier inom grön mjukvaruutveckling har visat att implementationer med designmönster kan påverka energikonsumtionen av en lösning både positivt och negativt [7], [12], [25]. I tabell 3 visas en sammanfattning av resultaten från studierna som beskrivs i följande stycken:

Tabell 3: Resultat från studier av Sahin el al. [7], Noureddine & Rajan [12] och Bunse & Stiemer [25]

Författare	**Designmönster**	**Energipåverkan (%)**
Sahin et al.	Decorator	712,89
	Observer	62,20
	Abstract Factory	36,47
	Flyweight	-58,08
	Proxy	-36,47
Noureddine & Rajan	Observer	30,63
	Mediator	26,61
	Decorator	12,24
Bunse & Stiemer	Decorator	133,60
	Prototype	32,20
	Abstract Factory	15,90

Sahin et al. [7] genomförde en studie där de mätte energikonsumtionen för inbyggda system med lösningar med och utan designmönster. De undersökte 15 designmönster, fem från varje kategori: skapande, strukturella och beteendemönster. Resultatet från studien visade att designmönster kan påverka energikonsumtionen både positivt och negativt och att kategorierna för designmönstren inte var en faktor i energikonsumtionen. De designmönster som visade på störst ökningar i energikonsumtion var: Decorator, Observer och Abstract Factory. De som minskade energikonsumtionen mest var Flyweight och Proxy.

Noureddine & Rajan [12] utgår i sin studie från resultat från Sahin et al. [7] och genomför mätningar på lösningar skrivna i C++ och Java [12]. Dessa uppvisar vissa skillnader i energikonsumtionspåverkan bland designmönstren där vissa resultat även skiljer sig från resultaten i studien av Sahin et al. [7]. Resultaten visar på stora ökningar av energikonsumtion för designmönstren: Observer, Mediator och Decorator. De undersöker sedan Observer och Decorator, för att se om de kan optimeras för att konsumera mindre energi. De valde bort Mediator med anledningen att det ska vara ett mindre populärt mönster med mycket gemensamt med Observer. Efter optimeringar av designmönstren påvisades resultat av minskad energikonsumtion på 4,32% – 25,47%.

I en studie av Bunse & Stiemer [25] genomförs mätningar på lösningar med och utan sex olika designmönster på tre mobila enheter. De jämför i dessa mätningar skillnaden mellan båda fallen i både exekveringstid och energikonsumtion. Resultaten påvisar skillnader i både energikonsumtion och exekveringstid vid användandet av designmönster. Tre av mönstren visade på en större ökning av energikonsumtion: Decorator, Prototype och Abstract Factory.

3 Metod och genomförande

Detta arbete ämnar svara på två frågeställningar varav en är kvalitativ och den andra är kvantitativ. För att svara på dessa användes mixad metod som ansats. Mixad metod definieras som en kombination av kvantitativa och kvalitativa metoder [26]. Genom att använda mixad metod minskar man sårbarheten som kommer med en enskild metod och ökar infallsvinklarna för insamling av data till en studie [26]. För denna studie var mixad metod ett självklart val för att svara på de två frågeställningarna:

- Hur ser utvecklare på Monitor på hållbarhet i förhållande till utvecklingsprocessen av system?
- Hur påverkas en mjukvarulösnings energikonsumtion av algoritmers tidskomplexitet och val av primitiva datatyper och datastrukturer?

Den kvalitativa frågeställningen om hur utvecklare på Monitor ser på hållbarhet undersöktes med hjälp av intervjuer som datainsamlingsmetod. Samtidigt som experiment användes för att svara på den kvantitativa frågeställningen om hur energikonsumtionen av en mjukvarulösning påverkas.

3.1 Datainsamling

Intervjuer används då man efterfrågar detaljerad information med flexibilitet i möjligheten till fördjupningar [27]. I denna studie skulle data samlas in om hur utvecklare på Monitor ser på hållbarhet under utvecklingsprocessen. Intervjuer valdes som datainsamlingsmetod för detta för att fånga upp fördjupningar och skapa möjligheten att ställa följdfrågor till respondenterna.

Som kvantitativ datainsamlingsmetod användes experiment. Experiment används för att undersöka hur en faktor påverkas under olika omständigheter [27]. Då resultaten som efterfrågades av experimentet var hur energikonsumtion påverkas av en algoritms tidskomplexitet och val av primitiva datatyper ansågs detta vara en lämplig metod för ändamålet.

3.1.1 Intervjuer

Semistrukturerade intervjuer användes för att samla in kvalitativa data om vad som påverkar val under utvecklingsprocessen för utvecklare på Monitor. En semistrukturerad intervju använder sig ofta av en intervjuguide som strukturerar frågor men lämnar utrymme för uppföljningar [27]. Då majoriteten av frågorna eftersökte ja- eller nej-svar med möjlighet till utveckling om varför så ansågs semistrukturerade intervjuer vara lämpliga som datainsamlingsmetod. Utveckling av svaren och möjligheten att ställa individuella följdfrågor var viktig för studien för att kunna måla upp en bild av hur utvecklingsprocessen såg ut på Monitor utan att veta något om den i början av studien.

Intervjufrågorna som kan läsas i bilaga A frågades i tre delar. Först undersöktes om energikonsumtion och miljöpåverkan är något som påverkar utvecklingsprocessen idag, samt vilka andra faktorer som påverkar respondenterna. Därefter ställdes frågor relaterade till standardisering och hur de påverkas av att arbeta i ett stort system med många utvecklare. Samtidigt ställdes frågor om hur de tänkte kring algoritmers tidskomplexitet och val av primitiva datatyper och datastrukturer. Till sist fick respondenterna svara på om de skulle önska att utveckla med fokus på energikonsumtion och miljöpåverkan, vilken sorts resurser de skulle önska ha för att kunna göra det och när under utvecklingsprocessen de ansåg att det borde tas in som en faktor. Denna struktur användes då det först var viktigt att veta vad respondenterna visste om grön mjukvaruutveckling, samt vad de ansåg var viktigt under utvecklingsprocessen. För att sedan kunna få svar på om de skulle vilja ta in grön mjukvaruutveckling i utvecklingsprocessen.

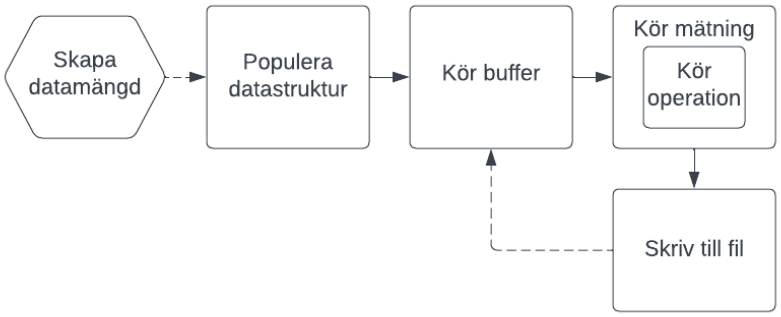
Då studien avser undersöka hur grön mjukvaruutveckling kan tillämpas på större kodbaser med många utvecklare, och Monitor är ett företag med många utvecklare i olika team, så passade utvecklare på Monitor bra som population för studien. Urvalet för intervjuerna ämnade upprätthålla reliabilitet (reliabilitet beskrivs närmare under rubrik 3.3.2 Reliabilitet) för studien. Krav ställdes därmed på respondenterna att i så stor utsträckning som möjligt vara av olika könsidentiteter, åldrar och erfarenhet som mjukvaruutvecklare. Dessutom valdes utvecklare från olika utvecklingsteam för att fortsatt upprätthålla reliabiliteten.

3.1.2 Experiment

Experimentet avsåg undersöka hur tidskomplexitet och val av primitiva datatyper och datastrukturer påverkar energikonsumtionen. För att ta reda på detta behövde data uppmätas för hur mycket energi som konsumerades vid olika omständigheter. För att kunna genomföra mätningar utvecklades en Javaapplikation. Java valdes som språk i och med författarens tidigare erfarenhet och att det är ett av de mest populära programmeringsspråken [28]. För applikationen togs beslut om vad som skulle mätas och under vilka omständigheter detta skulle göras. Applikationens krav följer nedan:

- Skapa datamängder av olika storlekar för både heltal och decimaltal och spara dessa till textfiler. Detta gjordes för att kunna återanvända samma datamängder vid alla mätningar.
- Tillhandahålla simpla datastrukturer.
- Läsa in en skapad datamängd från en textfil och fylla en datastruktur med värdena.
- Köra en buffer för att upprätthålla validiteten av energimätningarna.
- Starta en energimätning med RAPL vars resultat skrivs till en csv-fil.
- Köra önskad algoritm under energimätning i en loop 100 000 gånger för att uppmäta tillräckliga resultat.
- Skriva ut exekveringstiden för en mätning för användning i analysen.

Applikationens flöde visualiseras i figur 3.



Figur 3: Flödesdiagram för experimentapplikationen

Experimentets energimätningar genomfördes på en dator med specifikationerna i tabell 4. Datorn ominstallerades från Windows 10 till Ubuntu 22.04.2 LTS för att kunna använda RAPL som mätverktyg.

Tabell 4: Specifikationer för mätdatorn

Modell	Lenovo Lenovo YOGA 710-14IKB
Minne	8GiB
Processor	Intel® Core™ i7-7500U CPU @ 2.70GHz × 4
Grafikkort	Mesa Intel® HD Graphics 620 (KBL GT2)

För energimätningarna togs beslut för vad som skulle mätas och hur. Grunden för dessa beslut beskrivs nedan.

Algoritmerna som valdes behövde undersöka tidskomplexiteter, primitiva datatyper och datastrukturer och är samlade i bilaga B. För tidskomplexiteter och primitiva datatyper så skapades iterationer för att imitera växandet av en tidskomplexitet. Detta mättes i operation O i bilaga B. För varje iteration i operation O så hämtades ett värde ur en array då denna operation har en konstant tidskomplexitet, $O(1)$, och arrayer i Java kan hantera primitiva datatyper. Resterande operationer A–N i bilaga B mätte olika datastruktursoperationer för att se om energikonsumtionen påverkades av en datastrukturs uppbyggnad eller tidskomplexiteten av en operation.

För tidskomplexiteter togs beslutet att undersöka fem av de sju tidskomplexiteter som ofta återkommer i litteratur och artiklar och som är sammanställda i visualiseringen i figur 2: $O(1)$, $O(log\ n)$, $O(n)$, $O(n\ log\ n)$ och $O(n^2)$ [17]. $O(2^n)$ och $O(n!)$ valdes bort då dessa vid utförandet växte sig för stora vid nedan valda datamängder.

Storleken på datamängderna som skulle undersökas beslutades ta en främst exponentiellt växande form. Detta baserades på studierna om hur Javas datastrukturer påverkar energikonsumtionen. Där en av studierna mäter energikonsumtionen vid datamängder av storlekarna: 25 000, 250 000 och 1 000 000 värden [21] medan en annan uppmätte energin vid storlekarna: 32, 320, 3 200, 32 000, 320 000 och 3 200 000 värden [20]. Som ett mellanting så användes i denna studie datamängder av storlekarna: 1 000, 10 000, 100 000, 1 000 000 och 2 000 000 värden.

För att kunna testa hur storleken på primitiva datatyper påverkar energikonsumtionen så genomfördes mätningarna med hjälp av arrayer i tidskomplexitetsiterationer. I detta arbete undersöks Javas primitiva taldatatyper: byte, short, int, long, float och double. Storlekarna för dessa sträcker sig från 8 – 64 bitar som visas i tabell 5.

Tabell 5: Javas taldatatyper och deras storlekar [19]

Datatyp	**Storlek**
Byte	8 bitar
Short	16 bitar
Int	32 bitar
Long	64 bitar
Float	32 bitar
Double	64 bitar

Fem datastrukturer valdes ut att undersökas under experimentet: länkade listor, stackar, köer, mappar och set. Dessa valdes ut för sätten de lagrar data och referenser till datat för att kunna undersöka hur dessa referenser påverkar energikonsumtionen. De fem datastrukturerna kan delas in i tre grupper som visas i tabell 6. I dessa fall så implementerades länkade listor, stackar och köer på listnoder med antingen en eller två referenser till nästkommande eller föregående nod. Medan mappar och set implementeras med trädnoder med referenser till två nästkommande noder.

Tabell 6: Sammanfattning över hur datastrukturer implementeras för experimentet

Grupp	**Referenser**	**Skillnader inom grupp**	**Datastruktur**
Lista	Refererar till nästkommande nod	Referenser till start och slut med tillgång till alla	Länkad lista
		Referens till start och slut med tillgång till start	Kö
	Till föregående och nästkommande	Referens till slut med tillgång till slut	Stack
Hierarkisk	Refererar till två nästkommande noder	Unika nycklar med kopplade värden och tillgång till alla	Map
		Unika värden och tillgång till alla	Set

Då Javas bibliotek av datastrukturer består av längre hierarkier av olika implementationer så togs beslutet att implementera enkla versioner av de fem datastrukturerna. Detta gjordes för att fokusera experimentet på att undersöka datastrukturerna i grundkoncept och inte hur de blivit implementerade i Java. Javas implementation av arrayer användes då de inte har överflödiga metoder och arvshierarkier som de andra datastrukturerna. Noderna för datastrukturerna implementerades till att kunna hantera referenstypen Integer som värde.

Datastrukturerna definierades så att en länkad lista, stack och kö implementerades med hjälp av listnoder som håller i sitt värde och en referens till nästa listnod, varav stack även hade en referens till föregående nod. Skillnaderna mellan dem är att en stack endast kan komma åt senast tillagda noden, det vill säga toppen på stacken. En kö kan komma åt första och sista noden men bara ta bort den som var först in. Medan en länkad lista kan komma åt alla noder i listan från start till slut och lägga till och ta bort vilken nod som helst. Resterande datastrukturer: map och set definierades med hjälp av trädnoder som håller i sitt värde och två referenser till efterföljande trädnoder. Set och map kan endast innehålla unika värden. Skillnaden mellan dem är att i set lagras värdena rakt av medan i map så lagras en nyckel till ett sparat värde i trädstrukturen.

3.2 Dataanalys

Denna rubrik delas upp i två delar och beskriver hur dataanalysen gått till för de olika datainsamlingsmetoderna.

3.2.1 Intervjuanalys

Resultaten från intervjuerna bestod av två delar: inspelningar och intervjuformulär. Intervjuformulären summerades i en tabell i bilaga C. Inspelningarna transkriberades så ordagrant som möjligt för att kunna användas i en tematisk analys. Tematisk analys är ett sätt att analysera och presentera data genom att identifiera intressanta mönster, som kallas för teman [29]. Teman definieras som något intressant i relation till forskningsfrågan [29]. En tematisk analys genomförs i sex faser [29]:

1. I den första fasen bekantar man sig med materialet.
2. I den andra fasen skapas initiala koder som är mer dataspecifika än de teman som skapas.
3. I den tredje fasen söker man efter teman. I denna fas så tar man koderna från föregående fas och samlar ihop dem i möjliga teman.
4. I den fjärde fasen granskar man de teman som skapats och ser över om de är reella och hur väl data passar in i dem.

5. I den femte fasen definieras och namnges teman som är kvar efter granskning.
6. I den sjätte fasen skapas en rapport som beskriver datat utifrån de teman som skapats.

Fas ett började med transkribering av intervjuerna. När detta var klart lästes transkriberingarna igenom för att skapa en övergripande bild över datat. Därefter lästes de igen med målet att markera intressanta delar.

I fas två översattes de markerade delarna till initiala koder för att i fas tre kunna sammanfattas i teman. I dessa faser plockades koderna in i ett Exceldokument där varje respondent representerades av en rad och varje kod fick en kolumn. När teman eftersöktes i fas tre flyttades koderna runt och temanamn sattes som kolumnnamn och färgsattes. Om något data hade spår av flera teman lades de under båda kolumnerna och datacellen färgsattes med färgen för det andra temat. Detta gjordes för att i framtida faser kunna se hur de ursprungliga teman arbetade tillsammans.

I fas fyra granskades de teman som skapats mot både datat som lagts under dem och den totala datamängden, det vill säga de ursprungliga transkriberingarna. Vissa teman ansågs här vara för övergripande medan andra kunde kopplas ihop. De data som färgsatts med två teman undersöktes också här för att se var relationen mellan dem låg och huruvida dessa data behövde delas upp på något sätt, skapa ett nytt tema eller endast hörde hemma i ett tema.

Fas fem började med att skapa slutgiltiga namn för de teman som var kvar efter granskning vilket redovisas i resultatet för detta arbete under rubriker med de olika temans namn. Vilket ledde till fas sex där intressanta data för de olika teman tas upp under rubrikerna. Detta gjordes för att sammanställa resultaten från intervjuerna och svara på den kvalitativa forskningsfrågan om hur utvecklare på Monitor ser på hållbarhet i förhållande till utvecklingsprocessen.

3.2.2 Experimentanalys

Resultaten från experimenten bestod av csv-filer med mätdata. För att behandla dessa data användes MATLAB som är en programmeringsplattform särskilt lämpad för att analysera data [30]. I MATLAB lästes csv-filerna in som tabeller så att datat kunde användas i beräkningar och visualiseringar. Då det genomfördes ett stort antal mätningar som alla genererade stora mängder data så beslutades att diagramrepresentationer av datat var ett lämpligt sätt att visualisera datat för analys. I och med att diagram är ett sätt att få en överskådlig bild över hur data är fördelat [27].

För inläsningarna av datat till MATLAB analyserades csv-filerna i kombination med exekveringstiderna för varje algoritm. Detta gjordes för att endast läsa in de rader då en algoritm hade körts under mätningen med RAPL som körs under ett visst intervall. Raderna valdes ut så att den andra raden efter viloenergi fick vara med för visualiseringen. För att se till att ingen data missades så jämfördes raden mot exekveringstiden. Detta steg gjordes för att se till att visualiseringarna visade så korrekta data som möjligt.

Tre sorters visualiseringar skapades: 1) 3D-stapeldiagram över alla datastrukturs-mätningar. 2) Linjediagram för intressanta datastruktursmätningar. 3) Linjediagram för tidskomplexitetsmätningarna.

Det första diagrammet användes för att se hur datastrukturernas algoritmer påverkade energikonsumtionen vid olika datamängder. Det andra för att närmare jämföra skillnader mellan intressanta resultat från datastruktursmätningen. Det tredje användes för att visualisera tidskomplexitetsmätningarna med en linje för varje primitiv datatyp som använts för mätningen.

3.3 Validitet och Reliabilitet

Följande rubriker handlar om validitet och reliabilitet. Validitet handlar om giltighet [27]. För att uppnå validitet krävs att det som mätts under en studie var det som sattes ut att mätas [27]. Reliabilitet innebär att resultatet förblir detsamma vid upprepade mätningar [27]. I och med den mixade metoden så beskrivs validiteten och reliabiliteten nedan utifrån både ett kvantitativt och ett kvalitativt perspektiv.

3.3.1 Validitet

För att uppnå validitet i intervjuerna krävdes att det som undersökts svarar på frågeställningen: Hur ser utvecklare på Monitor på hållbarhet under utvecklingsprocessen? Detta gjordes delvis genom urvalet. Det största kravet som ställdes på urvalet var att respondenterna skulle vara utvecklare på Monitor. Därefter ställdes ytterligare krav att utvecklarna i så stor utsträckning som möjligt skulle vara av olika åldrar, erfarenhet som utvecklare, könsidentitet och vilka team de tillhörde på Monitor. Teamaspekten lades till så att eventuella skillnader i arbetssätt kunde fångas upp på ett annat sätt än om alla respondenter tillhörde ett team och hade arbetat fram ett gemensamt arbetssätt.

Förutom urvalet behövde intervjufrågorna undersöka frågeställningen för att uppnå validitet. Detta gjordes med hjälp av den struktur som beskrivs under rubriken 3.2.1 Intervjuanalys. Denna struktur menade undersöka hur utvecklare på Monitor ser på hållbarhet under utvecklingsprocessen innan intervjun och i framtiden.

För validitet i experimentet var första steget att ominstallera mätdatorn från att använda Windows till Ubuntu som operativsystem. Detta ledde till färre bakgrundsprocesser och en lägre basenergikonsumtion för datorn. Lägre basenergikonsumtion ledde till mindre risk att fel sak mättes på grund av fluktueringar i användning av energi av bakgrundsprocesser.

Vid varje mätning kördes sedan en buffer innan energimätningar av en algoritm. Detta gjordes för att låta datorns kylningssystem hinna göra sitt jobb innan mätningarna började och således påverka resultatet. Dessutom undveks att generering av datamängder påverkade energimätningarna då generering och påfyllning av datastrukturer genomfördes innan buffern för alla mätningar.

3.3.2 Reliabilitet

För den kvalitativa delen av denna studie kan reliabiliteten upprätthållas genom att se till att tillräckliga detaljer förmedlas om hur intervjuer har gått till för att kunna upprepas. I kombination med att studiekontexten, i detta fall Monitor, var en tillräckligt representativ population för att kunna representera hur utvecklare ser på hållbarhet i utvecklingsprocessen. Även urvalet på Monitor var viktigt för reliabiliteten. Urvalet behövde vara så spritt som möjligt över teamgränser, åldrar, könsidentiteter och erfarenhet för att resultaten skulle kunna överföras till andra studiekontexter.

Den kvantitativa reliabiliteten för experiment riskerades då man inte kan säkerställa att energikonsumtionen för mätdatorn var densamma som för en annan. Men då resultatet som efterfrågades handlar om relationer mellan energikonsumtion och olika gröna anpassningar så bör reliabiliteten kunna upprätthållas så länge mätningarna genomförs under samma förutsättningar.

3.4 Etiska överväganden

Under detta examensarbete har målet varit att följa de etiska riktlinjer som tagits upp av Säfsten & Gustavsson [27]. Deras riktlinjer är en sammanställning av olika riktlinjer och även lagar från olika institutioner och forskning. För att följa dessa har arbetet strävat efter att uppnå validitet och reliabilitet och resultera i ett ärligt och upprepbart resultat.

Etiska överväganden har främst varit aktuella för intervjuerna. Enligt Säfsten & Gustavsson [27] ska delar av en studie som hanterar människor göras med informerat samtycke och arbeta för att upprätthålla konfidentialitet. För att göra detta har resultaten i rapporten publicerats med största möjliga anonymitet och svar på urvalsfrågorna inte tagits upp i rapporten. Dessutom har minimal bearbetning gjorts under transkribering för att upprätthålla integriteten av respondenternas svar. De enda korrigeringar som gjordes var för att underlätta förståelse vid läsning utan att förlora innebörden.

4 Resultat

Resultatet delas upp enligt de två frågeställningarna för studien. Under 4.1 Utvecklares syn på hållbarhet, presenteras resultaten från intervjuerna och den tematiska analysen. Därefter summeras resultaten från experimentet med diagram ska-pade i MATLAB under 4.2 Påverkan på en mjukvarulösnings energikonsumtion.

4.1 Utvecklares syn på hållbarhet

Sju respondenter intervjuades för studien enligt frågorna i bilaga A. Två av sju respondenter identifierade sig som kvinnor och resterande fem som män. Sex av sju respondenter var i åldrarna 26–32, den yngsta var 26 år och den äldsta 61 år. Erfarenhetsfördelningen som utvecklare sträckte sig från 1 månad upp till 30 år varav alla förutom en respondent hade arbetat på Monitor under hela sin yrkesverksamma liv.

De summerade intervjuformulären återfinns i bilaga C. Resultaten från den tematiska analysen delades in i fem teman för att svara på den kvalitativa frågeställningen om hur utvecklare på Monitor ser på hållbarhet i förhållande till utvecklingsprocessen. Dessa teman är: förkunskaper, prestanda och tid, läsbarhet och standarder, testbarhet samt villighet och önskemål.

4.1.1 Förkunskaper

Förkunskaper omfattar respondenternas tidigare kunskap om grön mjukvaruutveckling och deras spontana tankar kring det. Den initiala frågan under intervjuerna frågade respondenterna om energikonsumtion och miljöpåverkan togs i åtanke under utvecklingsprocessen. Svaret för detta var av alla nej, med flera utvecklingar om att det var något man inte visste kunde vara en faktor eller aldrig hört talas om. I flera fall gjordes även spontana kopplingar till att energikonsumtionen borde kunna kopplas till prestandan och att man på så sätt indirekt hade energikonsumtion som faktor under utvecklingsprocessen. En av respondenterna svarade till exempel så här på frågan om miljöpåverkan och energikonsumtion tas i åtanke under utvecklingsprocessen: ”*Nej, nej inte alls. Alltså indirekt så är det väl det, men det är väl mest prestanda liksom som har fokus och inte miljöaspekter.*”

4.1.2 Prestanda och tid

Under intervjuerna återkom tid och prestanda i många av svaren av alla respondenter. Respondenterna svarade att prestanda var en påverkande och återkommande faktor för dem under utvecklingsprocessen. Vid frågor om man skulle välja en energisnål lösning eller en annan lösning så svarade flera respondenter att de skulle välja den lösning som var bäst när det kom till prestanda och tid. De svarade även att en energisnål lösning inte fick komma i vägen för en lösnings prestanda.

En av respondenterna tog även upp slutanvändarens upplevelse i koppling till prestanda och tid. De menade att om ett program tar lång tid att köras så kunde det vara ”stressande” för användaren och att undvika detta påverkade hur respondenten tänkte under utvecklingsprocessen.

4.1.3 Läsbarhet och standarder

Läsbarhet och standardisering ansågs vara en viktig del av kodskrivandet under utvecklingsprocessen. Kopplingar drogs till att Monitor är ett stort system med många utvecklare och att man är väldigt mån om att koden man skriver ska kunna förstås av så många som möjligt. En respondent nämnde kodstandarder med målet att öka kodkvaliteten på Monitor. I de flesta fallen valdes en standardiserad lösning före en energisnål lösning. Argumenten för standardiserade lösningar var att läsbarheten ökade möjligheten för att många utvecklare skulle kunna förstå kodbasen och arbeta i den. En av respondenterna resonerade så här i valet mellan en energisnål lösning och en standardiserad lösning:

> *”Ja, det beror på standardsvaret. Men när man tänker i en produkt som Monitor jobbar med. Då tror jag att värdet att ha det standardiserat är väldigt högt eftersom att det är många andra som kommer behöva underhålla det och så vidare och det är en väldigt stor kodbas […] så jag skulle nog ändå luta mig åt [standardiserade] hållet om det inte var en enorm skillnad liksom.”*

4.1.4 Testbarhet

Två av respondenterna nämnde testbarhet som en viktig faktor i utvecklingsprocessen. Båda dessa respondenter nämner att användandet av olika designmönster görs för att underlätta testningen. Testbarheten sattes även som krav vid valet av en standardiserad lösning mot en energisnål lösning med utvecklingen att man ofta sitter halva tiden med att skriva tester för funktionalitet och om testerna blir svåra att skriva för en lösning så väljs denna ofta bort mot en mer testbar lösning.

4.1.5 Villighet och önskemål

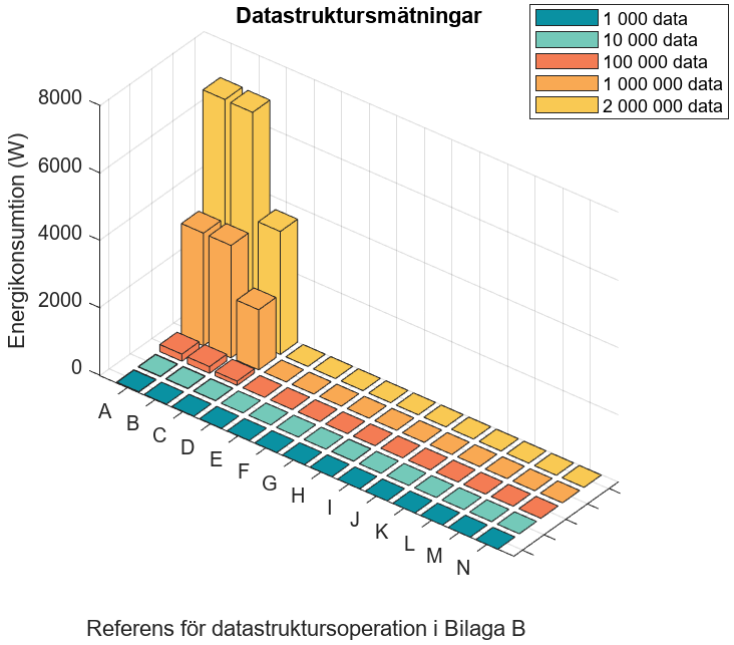
Villighet omfattar svar om framtida involvering av grön mjukvaruutveckling i utvecklingsprocessen. Det vill säga hur respondenterna har svarat vid val av energisnåla lösningar jämte standardiserade lösningar och hur de svarat på frågan om de skulle vilja ta in energikonsumtion och miljöpåverkan som en faktor i utvecklingsprocessen i framtiden. Önskemål berör svar om hur respondenterna anser att vägen framåt med grön mjukvaruutveckling kan se ut.

Majoriteten av respondenterna svarade att de i framtiden skulle vilja ta in energikonsumtion och miljöpåverkan som en faktor i utvecklingsprocessen. De som inte ville det kopplade till att resultatet borde bli detsamma oavsett så länge man fokuserade på prestanda och att de hade andra högre prioriterade arbetsuppgifter.

För att kunna utveckla med fokus på energikonsumtion och miljöpåverkan ställdes frågor om när i utvecklingsprocessen respondenterna tyckte man skulle ta in det. Alla respondenter ansåg att det borde tas in så tidigt som möjligt och vara med under hela processen. Frågan ställdes även om vad respondenterna trodde att de skulle behöva för att kunna ta in energikonsumtion och miljöpåverkan som aspekt. Majoriteten svarade att de vill ha mer information för att veta vad som påverkar energikonsumtionen och hur mycket det påverkar. En av respondenterna sade att ”*information tycker jag är ett bättre sätt för jag då skulle kunna se själv hur det är tänkt och fungera*” men att söka efter informationen kunde vara svår gällande att veta att informationen är pålitlig och bra. Utöver information så önskades även verktyg för att se hur energikonsumtionen påverkades av olika val, till exempel direkt i metodblocket.

4.2 Påverkan på en mjukvarulösnings energikonsumtion

Resultaten från experimentet visade en konstant energiförbrukning på 9–11 Watt varje sekund som en programinstruktion exekverades. Den totala energiförbrukningen var ett resultat av den konstanta energikonsumtionen, exekveringstiden för en programinstruktion och tidskomplexiteten av en algoritm.



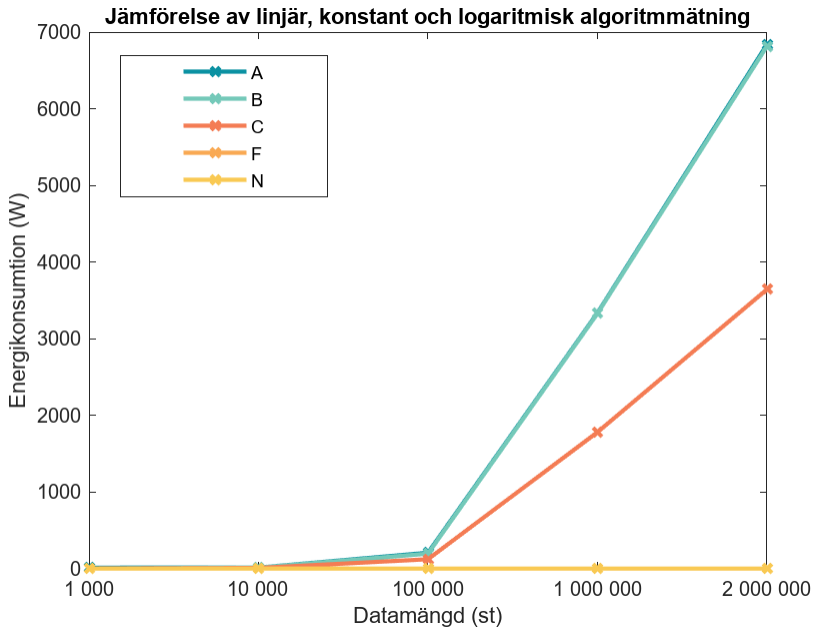
Figur 4: 3D-stapeldiagram över datastruktursmätningar för alla datamängder

I figur 4 visualiseras alla datastruktursoperationer vid alla datamängder. Majoriteten av operationerna konsumerade lika mycket energi oavsett datamängd och datastruktursuppbyggnad. Detta gällde oavsett en operations tidskomplexitet. Dessa operationer exekverades på under en sekund oavsett datamängd vilket resulterade i en energikonsumtion på max 11 Watt baserat på den konstanta energiförbrukningen. Tre operationer sticker ut: A, B och C. Dessa operationer har alla en linjär tidskomplexitet, $O(n)$, och beskrivs i tabell 7.

Tabell 7: Beskrivningar för utvalda datastruktursoperationer taget från Bilaga B

	Datastruktursoperation	**Tidskomplexitet**
A	Lägga till och ta bort näst sista värdet i en länkad lista	$O(n)$
B	Hämta näst sista värdet i en länkad lista	$O(n)$
C	Lägga till och ta bort sista värdet i en länkad lista	$O(n)$
F	Hämta första värdet i en länkad lista	$O(1)$
N	Hämta värde ur en map	$\theta(log\ n)/\ O(n)$

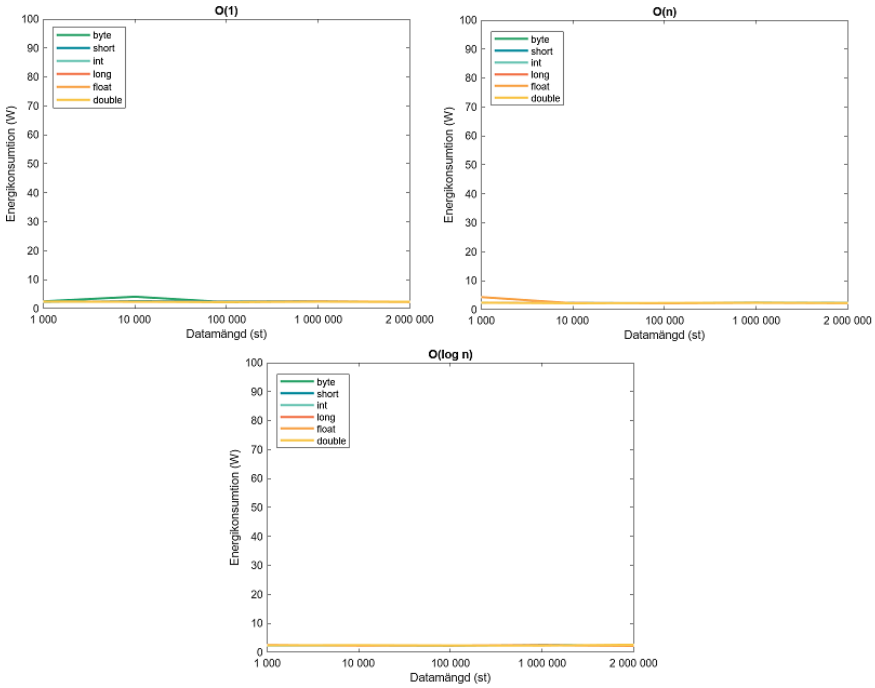
Operation C skiljer sig från A och B då att lägga till sista värdet har en konstant tidskomplexitet, $O(1)$, medan att ta bort det har en linjär tidskomplexitet, $O(n)$. Kombinationen resulterade i en linjär tidskomplexitet med en dubbelt så snabb exekveringstid jämfört med A och B. I figur 4 och 5 kan man se att energikonsumtionen för att lägga till och ta bort sista värdet i en länkad lista ligger på runt hälften av energikonsumtionen för de andra två operationerna i och med den halverade exekveringstiden.



Figur 5: Jämförelse av linjär, konstant och logaritmisk algoritmmätning

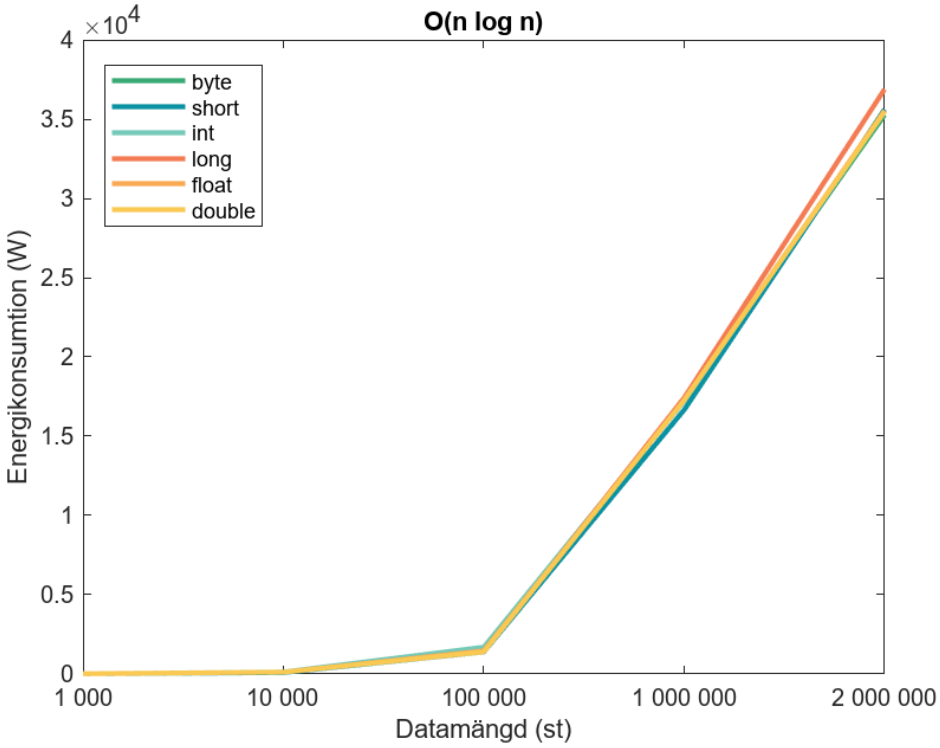
I figur 5 visualiseras de tre operationer som stod ut under datastruktursmätningarna, A, B och C, samt operationerna F och N som representerar mätningar för operationer med konstant, $O(1)$, respektive logaritmisk tidskomplexitet, $\Theta(log\ n)$. Dessa operationer beskrivs i tabell 7. I detta fall uppmättes inga skillnader mellan de konstanta och de logaritmiska operationerna. Detta berodde på att även vid största datamängden 2 000 000 så innebär det endast cirka 21 iterationer för en logaritmisk tidskomplexitet.

Figur 6–8 visualiserar resultaten av tidskomplexitetsmätningarna. I figur 6 visualiseras mätningarna för konstant, logaritmisk och linjär tidskomplexitet. Dessa mätningar uppmätte ingen skillnad i energikonsumtion. Exekveringstiderna per iteration var snabba och alla mätningar var klara på under en sekund oavsett datamängd. Energikonsumtionen resulterade därmed i samma resultat som majoriteten av mätningarna i datastruktursmätningarna i figur 4.



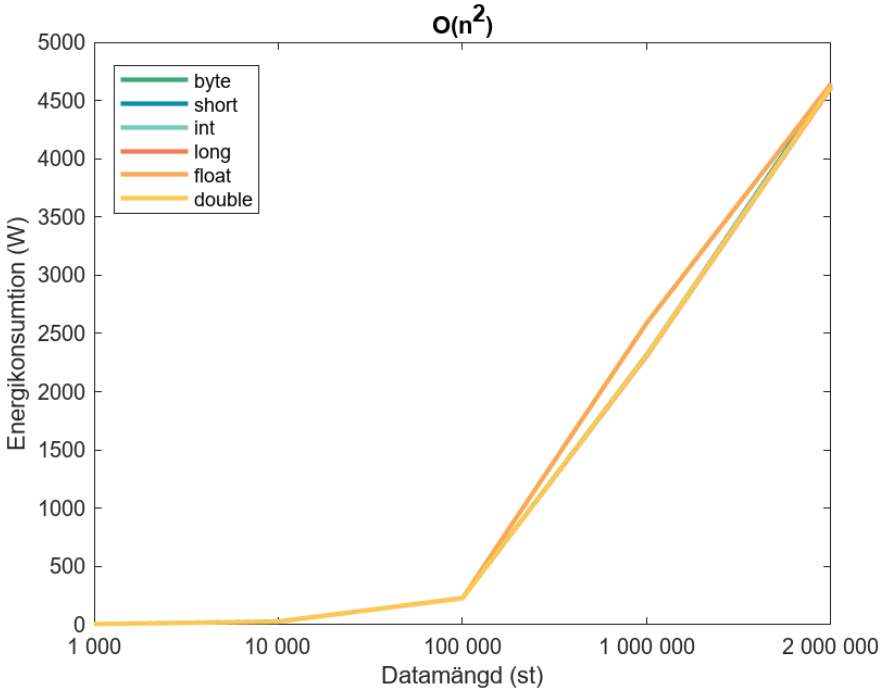
Figur 6: Konstant tidskomplexitet $O(1)$ (Vänster), Logaritmisk tidskomplexitet $O(\log n)$ (Mitten) och Linjär tidskomplexitet $O(n)$ (Höger)

I figur 7 visualiseras resultaten av mätningarna på en iteration med linjärlogaritmisk tidskomplexitet, $O(n \log n)$. Denna operation hade längst exekveringstid och därmed störst energikonsumtion då mätningen med datamängden 2 000 000 data tog runt en timme att genomföra.



Figur 7: Linjärlogaritmisk tidskomplexitet $O(n \log n)$

Figur 8 visualiserar mätningarna över kvadratisk tidskomplexitet, $O(n^2)$. Denna visar ett liknande resultat som det i figur 7 men lägre då exekveringstiden per iteration var lägre. Detta ledde till en kortare tid och lägre energikonsumtion trots att antalet iterationer var högre. Skillnaden i exekveringstid kom från att den kvadratiska operationen endast ökade en inre loop med en för varje iteration. Medan den linjärlogaritmiska operationen dubblerade sitt index för den inre loopen varje iteration vilket visade sig vara en operation med betydligt längre exekveringstid.



Figur 8: Kvadratisk tidskomplexitet $O(n^2)$

5 Diskussion

Diskussionen för detta arbete delas upp i två delar, metoddiskussion och resultatdiskussion. Metoddiskussionen diskuterar hur olika val under intervjuerna och experimentet kan ha påverkat resultaten. Medan resultatdiskussionen omfattar resultaten från både den kvalitativa och den kvantitativa delen och mynnar ut i att kombinera dessa resultat och svara på hur grön mjukvaruutveckling kan ta plats i standardiseringen i system med många utvecklare.

5.1 Metoddiskussion

Metoddiskussionen delas upp i två delar och diskuterar först hur intervjuer påverkat resultaten för att sedan diskutera hur val under experimentet påverkat resultaten.

5.1.1 Intervjuer

Intervjuer användes i denna studie då den kvalitativa frågeställningen handlade om hur en utvecklare på Monitor ser på hållbarhet under utvecklingsprocessen. Enkäter hade kunnat användas med samma ändamål och med ett mer kvantitativt resultat men valdes bort för att inte missa följdfrågor och utvecklingar av svar. Resultatet av att använda enkäter hade dock kunnat vara mer representativt till följd av att enkäter lättare hade kunnat undersöka en större andel av populationen.

Frågorna som ställdes under intervjuerna var delvis riktade mot val som undersöktes under experimentet. Resultaten från intervjuerna skulle ha kunnat påverkas om den influensen inte existerade. Detta hade kunnat resultera i ett resultat som var mer fokuserat på utvecklingsprocessen i stort i stället för på specifika utvecklingsval.

5.1.2 Experiment

Beslutet att använda RAPL som energimätningsverktyg togs tidigt under arbetets gång. RAPL har gett tydliga resultat men en bristande funktion var hur man behövde sätta hur länge den mäter i stället för att kunna starta mätningen i början av en algoritm och avsluta den när algoritmen var klar, vilket hade kunnat ge tydligare resultat. Särskilt då exekveringstiden för en algoritm var antingen väldigt mycket kortare eller längre än mätfönstret. För att motverka dessa brister skrevs exekveringstiden för varje algoritm ut och sparades i textfiler för jämförelse under resultatet.

Java valdes som programmeringsspråk då kunskapen redan fanns hos författaren i kombination med att det är ett av de mest använda programmeringsspråken [26]. Ett annat språk hade kunnat bidra med annan funktionalitet och andra möjligheter till energibesparingar men då arbetet utfördes av en person på under tio veckor så fokuserades tiden på andra delar.

Valet gjordes att testa energikonsumtionen av olika val isolerat. Detta kan ha påverkat resultaten på flera sätt. Man kan vara säker på validiteten av en mätning samt att man mäter just datatypen, datastrukturen eller tidskomplexiteten. Men man ser det inte i ett sammanhang av en faktisk mjukvarulösning. Detta kan vara både positivt och negativt då man i en faktisk mjukvarulösning skulle kunna se hur andra val påverkar energikonsumtionen tillsammans med de val som undersökts här vilket hade kunnat ge intressanta insikter. Det skulle dock drabba validiteten för just denna studie.

5.2 Resultatdiskussion

Resultaten från intervjuerna och den tematiska analysen visar att det finns en villighet att utveckla med fokus på grön mjukvaruutveckling men att kunskapen om det inte finns. Det största behovet anses vara information. Respondenterna anser att med tillräcklig information om hur olika val påverkar energikonsumtionen så kan man sedan ha det i åtanke under utvecklingsprocessen. De säger även att man bör ta in det som en aspekt redan från planeringen för att ha med det i åtanke genom hela processen.

Information om gröna val kan spridas med hjälp av bland annat de studier som tagits upp under bakgrunden för denna studie. Som en av respondenterna beskrev kan det dock vara svårt att sålla bland studier och information och veta hur pålitlig informationen är. Information om grön mjukvaruutveckling bör därmed sammanställas och göras lättillgänglig till utvecklare för att kunna användas under utvecklingsprocessen samt för att skapa gröna alternativ till kodstandarder.

Experimentet i denna studie bidrar med resultatet att energikonsumtionen är direkt kopplad till exekveringstiden av programinstruktioner och en algoritms tidskomplexitet. Denna kombination resulterar i prestanda och innebär att det fokus som respondenterna svarat att de hade kan ses som ett sätt att indirekt ta in hållbarhet i utvecklingsprocessen.

Experimentet har inte undersökt alla faktorer i koden som kan påverka energikonsumtionen. Men det har visat att primitiva datatyper, åtminstone i Java, inte har en märkbar påverkan på energikonsumtionen för datamängder upp till 2 000 000. För datastrukturer visade experimentet att den påverkande faktorn för enkla datastrukturer var hur tidskomplexiteten för datastrukturens operationer såg ut och inte hur referenserna mellan noderna såg ut.

Resultaten har visat att standardiseringar är en viktig del av utvecklingsarbetet på Monitor och att designmönster används för att upprätthålla standardisering, läsbarhet och även testbarhet. Under intervjuerna fick respondenterna välja mellan standardiserade lösningar och energisnåla lösningar. Resultatet visade att majoriteten skulle välja en standardiserad lösning. De energisnåla lösningarna i dessa scenarion var inte standardiserade.

Med resultaten från experimentet som kopplat energikonsumtionen av en mjukvarulösning till exekveringstid och tidskomplexitet kan man dra slutsatsen att gröna mjukvarulösningar inte behöver vara lika individuella som antagits under introduktionen. Om energisnåla lösningar inte behöver vara individuella kan gröna standarder skapas för att minska en mjukvarulösnings energikonsumtion medan man bibehåller att standarder används i system med många utvecklare som även rekommenderas av Martin [23].

Kopplingen mellan designmönster och standardiseringar visar även att kodstandarder kan implementeras med hjälp av resultaten från studierna av Sahin et al. [7], Noureddine & Rajan [12] och Bunse & Stiemer [25]. Informationen från dessa studier om hur olika designmönster påverkar en mjukvarulösnings energikonsumtion ger en indikation på hur man kan göra mer eller mindre gröna val med designmönster. Medan studien av Noureddine & Rajan [12] även kan användas för att skapa gröna optimeringar av designmönster i skapandet av nya gröna standarder.

Med resultatet att simpla datastrukturers energipåverkan för datamängder upp till 2 000 000 data endast påverkades av operationernas algoritmer, så kan slutsatsen dras att man vid valet av datastrukturer bör tänka över vilka operationer som kommer behöva hanteras av datastrukturen. För att skapa gröna standarder för datastrukturer bör man kunna utgå från en datastrukturs baskoncept. Dock bör olika programmeringsspråks datastrukturer undersökas såsom Javas har studerats i studierna av Pinto et al. [20], Pereira et al. [21] och Hasan et al. [22] för att se om andra faktorer påverkar energikonsumtionen för dessa. Genom att studera ett språks datastrukturer bör man sedan kunna sammanfatta vilka datastrukturer som är mer eller mindre gröna val.

5.3 Aspekter på miljö och hållbar utveckling

Miljö och hållbar utveckling var högst aktuellt i detta arbete då syftet med studien var att främja ett mjukvarufokus för minskad energikonsumtion i system med många utvecklare.

FN:s 17 globala mål för en bättre värld antogs 2015 [31]. För detta arbete kan direkta kopplingar göras till åtminstone två av de globala målen. Mål 9: ”Hållbar industri, innovationer och infrastruktur” och mål 12: ”Hållbar konsumtion och produktion” [31]. Eftersom studiekontext för denna studie var Monitor, vars affärssystem är riktat mot tillverkande företag, kan kopplingar göras till dessa mål.

Resultatet från studien säger att energikonsumtion kan minskas för system genom att fokusera på prestanda och gröna val. Att ta in dessa tankar i ett system såsom Monitor som används på över 5 000 företag skulle därmed kunna hjälpa i målen att nå en större hållbarhet i tillverkande företag och i arbetet att nå mål 9 och mål 12.

6 Slutsatser

Resultaten visar att utvecklare på Monitors syn på hållbarhet under utvecklingsprocessen inte var en faktor då de inte visste att det kunde vara en del av den. Den viktigaste riktlinjen för utvecklarna var prestanda vilket kan kopplas till experimentresultaten som visar att energikonsumtionen är fullt beroende av exekveringstiden av en programinstruktion och en algoritms tidskomplexitet. Experimentet har även visat att val av primitiva datatyper inte påverkar en lösnings energikonsumtion. Samt att en datastrukturs påverkan beror på dess operationers algoritmer.

Denna studie visar att energikonsumtionen under exekveringstid är konstant oavsett programinstruktion och att en utvecklare påverkar energikonsumtionen genom att fokusera på prestanda. Resultaten har därmed visat att antagandet om att en minskad energikonsumtion för mjukvarulösningar kräver individuella gröna val är felaktigt, åtminstone när det kommer till primitiva datatyper och datastrukturer, och att det med prestanda kan standardiseras.

Slutsatsen för det här examensarbetet är att för att ta in grön mjukvaruutveckling i system med standarder så behöver nya gröna standarder skapas. För att skapa dessa standarder bör fokus läggas på prestanda och att sammanställa gröna val för lättare tillgång för utvecklare. Det finns stor potential för att sänka energikonsumtionen i mjukvarulösningar genom att öppna utvecklares ögon för möjligheten att minska ett systems energikonsumtion med konkreta gröna val under utvecklingsprocessen.

6.1 Framtida forskning

Framtida forskning bör läggas på att undersöka hur detaljer i kod påverkar energikonsumtionen för riktiga mjukvarulösningar. En möjlig studie skulle kunna vara att utveckla verktyg som kan användas av utvecklare under utvecklingsprocessen för att jämföra hur energikonsumtionen påverkas när val görs. Dessa resultat bör sedan observeras och användas för att identifiera hur val under utvecklingsprocessen påverkar energikonsumtionen i faktiska mjukvarulösningar, eftersom denna studie endast undersökt isolerade val.

Framtida forskning bör även arbeta för att samla information om specifika gröna val som kan implementeras av utvecklare och utvecklingsteam. Informationen bör samlas och göras lättillgänglig för att öka implementationen av grön mjukvaruutveckling. Detta kan göras delvis genom att studera datastrukturer och andra komponenter och verktyg i olika programmeringsspråk såsom studier genomförts på datastrukturer i Java. Det skulle ge utvecklare en bra överblick för hur de kan utveckla grönt i sina egna miljöer med konkreta exempel. Målet med dessa studier bör vara att sprida grön mjukvaruutveckling från forskningskretsar till arbetande utvecklare.

Referenser

[1] ”Data Centres and Data Transmission Networks – Analysis - IEA”. https://www.iea.org/reports/data-centres-and-data-transmission-networks (åtkomstdatum 15 mars 2023).

[2] ”Aviation – Analysis - IEA”. https://www.iea.org/reports/aviation (åtkomstdatum 16 mars 2023).

[3] E. Kern, M. Dick, S. Naumann, och T. Hiller, ”Impacts of software and its engineering on the carbon footprint of ICT”, *Environ Impact Assess Rev*, vol. 52, s. 53–61, apr. 2015, doi: 10.1016/j.eiar.2014.07.003.

[4] G. Procaccianti, H. Fernández, och P. Lago, ”Empirical evaluation of two best practices for energy-efficient software development”, *Journal of Systems and Software*, vol. 117, s. 185–198, juli 2016, doi: 10.1016/J.JSS.2016.02.035.

[5] ”Framtidens affärssystem | Monitor ERP | För tillverkande företag”. https://www.monitorerp.com/sv/erp-system/ (åtkomstdatum 02 maj 2023).

[6] J. Michanan, R. Dewri, och M. J. Rutherford, ”GreenC5: An adaptive, energy-aware collection for green software development”, *Sustainable Computing: Informatics and Systems*, vol. 13, s. 42–60, mar. 2017, doi: 10.1016/j.suscom.2016.11.004.

[7] C. Sahin *m.fl.*, ”Initial explorations on design pattern energy usage”, *2012 1st International Workshop on Green and Sustainable Software, GREENS 2012 - Proceedings*, s. 55–61, 2012, doi: 10.1109/GREENS.2012.6224257.

[8] S. Georgiou, S. Rizou, och D. Spinellis, ”Software Development Lifecycle for Energy Efficiency”, *ACM Computing Surveys (CSUR)*, vol. 52, nr 4, s. 81, aug. 2019, doi: 10.1145/3337773.

[9] ”Running Average Power Limit Energy Reporting CVE-2020-8694,...” https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html (åtkomstdatum 26 april 2023).

[10] ”tools/power/rapl — Firefox Source Docs documentation”. https://firefox-source-docs.mozilla.org/performance/tools_power_rapl.html (åtkomstdatum 26 april 2023).

[11] T. A. Ghaleb, ”Software energy measurement at different levels of granularity”, *2019 International Conference on Computer and Information Sciences, ICCIS 2019*, maj 2019, doi: 10.1109/ICCISCI.2019.8716456.

[12] A. Noureddine och A. Rajan, ”Optimising Energy Consumption of Design Patterns”, i *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, aug. 2015, s. 623–626. doi: 10.1109/ICSE.2015.208.

[13] E. Jagroep *m.fl.*, ”Awakening awareness on energy consumption in software engineering”, *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Society Track, ICSE-SEIS 2017*, s. 76–85, juni 2017, doi: 10.1109/ICSE-SEIS.2017.10.

[14] J. G. Brookshear och D. Brylow, *Computer science : an overview*, 13:e uppl. Pearson Education, 2019.

[15] M. A. Weiss, *Data structures and problem solving using Java*, 4:e uppl. Pearson Education, 2013.

[16] S. Gayathri Devi, K. Selvam, och S. P. Rajagopalan, ”An abstract to calculate big o factors of time and space complexity of machine code”, *IET Conference Publications*, vol. 2011, nr 583 CP, s. 844–847, 2011, doi: 10.1049/CP.2011.0483.

[17] ”Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell”. https://www.bigocheatsheet.com/ (åtkomstdatum 25 april 2023).

[18] ”Chapter 4. Types, Values, and Variables”. https://docs.oracle.com/javase/specs/jls/se12/html/jls-4.html (åtkomstdatum 02 maj 2023).

[19] ”Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics)”. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html (åtkomstdatum 30 mars 2023).

[20] G. Pinto, K. Liu, F. Castor, och Y. D. Liu, ”A comprehensive study on the energy efficiency of Java’s thread-safe collections”, *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, s. 20–31, jan. 2017, doi: 10.1109/ICSME.2016.34.

[21] R. Pereira, M. Couto, J. Saraiva, J. Cunha, och J. P. Fernandes, ”The influence of the Java collection framework on overall energy consumption”, *Proceedings - International Conference on Software Engineering*, s. 15–21, maj 2016, doi: 10.1145/2896967.2896968.

[22] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, och A. Hindle, ”Energy profiles of Java collections classes”, *Proceedings - International Conference on Software Engineering*, vol. 14-22-May-2016, s. 225–236, maj 2016, doi: 10.1145/2884781.2884869.

[23] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1:a uppl. USA: Prentice Hall PTR, 2008.

[24] E. Gamma, R. Helm, R. Johnson, och J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis: Pearson Education Corporate Sales Division, 1994.

[25] C. Bunse och S. Stiemer, ”On the Energy Consumption of Design Patterns”, *Softwaretechnik-Trends*, vol. 33, nr 2, s. 4–5, maj 2013, doi: 10.1007/S40568-013-0020-6.

[26] J. Heigham och R. A. Croker, ”Qualitative research in applied linguistics: A practical introduction”, *Qualitative Research in Applied Linguistics: A Practical Introduction*, s. 1–329, juni 2009, doi: 10.1057/9780230239517/COVER.

[27] Kristina. Säfsten och Maria. Gustavsson, *Forskningsmetodik : för ingenjörer och andra problemlösare*, Upplaga 1. 2019.

[28] ”The top programming languages | The State of the Octoverse”. https://octoverse.github.com/2022/top-programming-languages (åtkomstdatum 21 maj 2023).

[29] V. Braun och V. Clarke, ”Using thematic analysis in psychology”, *Qual Res Psychol*, vol. 3, nr 2, s. 77–101, 2006, doi: 10.1191/1478088706QP063OA.

[30] ”What Is MATLAB? - MATLAB & Simulink”. https://se.mathworks.com/discovery/what-is-matlab.html (åtkomstdatum 27 april 2023).

[31] ”Globala målen – Läs om Globala målen – 17 mål för hållbar utveckling”. https://www.globalamalen.se/om-globala-malen/ (åtkomstdatum 16 mars 2023).

Bilaga A: Intervjufrågor

Frågor om urval

- Ålder?
- Könsidentitet?
- Hur länge har du arbetat med mjukvaruutveckling?
- Hur länge har du arbetat som utvecklare på Monitor?

Frågor om vad som påverkar utvecklingsprocessen

- Är miljöpåverkan och energikonsumtion något du har i åtanke när du utvecklar idag?
    - Om ja: påverkar det hur du utvecklar?
        - Om ja: på vilket/vilka sätt?
    - Om nej: varför inte?
- Påverkas du av andra saker när du utvecklar, till exempel prestanda, minneshantering, kodkvalité, standardisering…?

Frågor om utvecklingsval

- Om du väljer en datatyp för en variabel funderar du då över hur stora värden den ska hantera?
- Nedgraderar du datatypen om variabeln inte kommer hantera tillräckligt stora värden?
- Om nedgradering av datatyp skulle innebära att den skulle stå ut mot andra variabler skulle du ändå nedgradera den?
- Använder du gärna designmönster?
    - Om ja, varför och vilka?
        - Om ett designmönster skulle påverka energikonsumtionen negativt skulle du välja bort det då?
    - Om nej, varför inte?
- Är tidskomplexitet något du tänker över under utvecklingsprocessen?
    - Om ja:

        - I koppling till datastrukturer?
        - I koppling till algoritmer?
- Om valet står mellan en energisnål implementation och en huvudsakligen standardiserad implementation, till exempel med ett designmönster. Vilken variant väljer du då?
- Skulle dina svar ovan påverkas om du skulle göra samma val i ett mindre hobbyprojekt gentemot i Monitors större system med många utvecklare?

Frågor om framtida miljömedveten utveckling

- Skulle du vilja utveckla med större fokus på energikonsumtion och miljöpåverkan?
    - Om nej: varför inte?
    - Om ja:
        - När i utvecklingsprocessen anser du att det skulle passa bäst att ta in energikonsumtion som faktor?
        - Finns det nått du skulle önska för att kunna utveckla mer miljömedvetet?
            - Utbildning, verktyg, undansatt tid…?

Bilaga B: Algoritmer för energimätning

	Beskrivning	Tidskomplexitet
A	Lägga till och ta bort näst sista värdet i en länkad lista	$O(n)$
B	Hämta näst sista värdet i en länkad lista	$O(n)$
C	Lägga till och ta bort sista värdet i en länkad lista	$O(n)$
D	Hämta sista värdet i en länkad lista	$O(1)$
E	Lägga till och ta bort första värdet i en länkad lista	$O(1)$
F	Hämta första värdet i en länkad lista	$O(1)$
G	Lägga till och ta bort ett värde i en stack	$O(1)$
H	Hämta värde i en stack	$O(1)$
I	Lägga till och ta bort ett värde i en kö	$O(1)$
J	Hämta värde i en kö	$O(1)$
K	Lägga till och ta bort värde ur ett set	$\theta\ (log\ n)/O(n)$
L	Hämta värde ur ett set	$\theta\ (log\ n)/O(n)$
M	Lägga till och ta bort värde ur en map	$\theta\ (log\ n)/O(n)$
N	Hämta värde ur en map	$\theta\ (log\ n)/O(n)$
O	Tidskomplexitetsiterationer	$O(1)/O(log\ n)/$ $O(n)/O(n\ log\ n)/$ $O(n2)$

Bilaga C: Summering av intervjuformulär

	R1	R2	R3	R4	R5	R6	R7
Är miljöpåverkan och energikonsumtion något du har i åtanke när du utvecklar?	Nej	Nej	Nej	Nej	Nej	Nej	Nej
Om du väljer en datatyp för en variabel funderar du då över hur stora värden den ska hantera?	Ja	Nej	Ja	Ja	Nej	Nej	Nej
Nedgraderar du datatypen om variabeln inte kommer hantera tillräckligt stora värden?	Ja	Nej	Ja	Nej	Nej	Nej	Nej
Om nedgradering av datatyp skulle innebära att den skulle stå ut mot andra variabler skulle du ändå nedgradera den?	Ja	Nej	Nej	Nej	Nej	Nej	Ja
Använder du gärna designmönster?	Ja	Nej	Ja	Ja	Ja	Ja	Ja
Om ett designmönster skulle påverka energikonsumtionen negativt skulle du välja bort det då?	Ja	Nej	Nej	Ja	Nej	Nej	Nej
Är tidskomplexitet något du tänker över under utvecklingsprocessen?							
- Datastrukturer	Nej	Nej	Ja	Ja	Ja	Ja	Ja
- Algoritmer	Nej	Ja	Ja	Ja	Ja	Ja	Ja
Om valet står mellan en energisnål implementation och en huvudsakligen standardiserad implementation, till exempel med ett designmönster. Vilken variant väljer du då?							
- Energisnål lösning	Nej	Ja	Nej	Ja	Nej	Nej	Nej
- Standardiserad lösning	Ja	Nej	Ja	Nej	Ja	Ja	Ja
Skulle dina svar ovan påverkas om du skulle göra samma val i ett mindre hobbyprojekt gentemot i Monitors större system med många utvecklare?	Ja	Nej	Ja	Nej	Nej	Ja	Nej
Skulle du vilja utveckla med större fokus på energikonsumtion och miljöpåverkan?	Ja	Ja	Ja	Nej	Ja	Ja	Nej