Ett system för kvalitativ evaluering av kodkvalitet

Analys av kodmönster med metaramverket Nuxt implementerade för spelet *Taloussankari Junior*

Martin Fellman

Examensarbete
Online Media
2022

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Online media
Identifikationsnummer:	8676
Författare:	Martin Fellman
Arbetets namn:	Ett system för kvalitativ evaluering av kodkvalitet – Analys av kodmönster med metaramverket Nuxt implementerade för spelet Taloussankari Junior
Handledare (Arcada):	Mirko Ahonen
Uppdragsgivare:	-

Sammandrag:
I detta arbete försöker skribenten utforma ett system för att evaluera kodkvalitet som sedan används för att evaluera ett antal kodmönster, kallade ”pause()”, ”kombination av reaktiviteten i Vue och både anpassade egenskaper och calc() i CSS” och ”central hantering av programtillstånd med Vuex”. Kodmönstren är plockade ur spelet Taloussankari Junior av Talous ja nuoret TAT och 10Monkeys, i vilket skribenten var så gott som helt självständig som programmerare.

Systemet baseras på litteratur av bl.a. McConnell (2004), Gamma m.fl. (1995), Brown m.fl. (1998) samt Internationella standardiseringsorganisationen (2011). Urvalet av kodmönster baseras på skribentens erfarenhet med koden för Taloussankari Junior.

I arbetet konstateras det att särskilt ”central hantering av programtillstånd med Vuex” är ett mönster som håller hög standard enligt systemet, medan ”pause()” under vissa omständigheter är väldigt problematiskt. Systemet som används för evalueringen är ändå bristfälligt då det appliceras enligt skribentens metod, så resultaten kan inte anses pålitliga.

Evalueringssystemet spekuleras ändå kunna användas med framgång i diagnostiska syften som del av en programmerares kritiska inställning till den kod hen arbetar med.

Nyckelord:	JavaScript, Nuxt, Vue, kodkvalitet, webbutveckling, programmering
Sidantal:	56
Språk:	svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Online Media
Identification number:	8676
Author:	Martin Fellman
Title:	A System for Qualitative Evaluation of Code Quality – Analysis of Programming Patterns under the Meta-Framework Nuxt, Implemented for the Game Taloussankari Junior
Supervisor (Arcada):	Mirko Ahonen
Commissioned by:	-

Abstract:
In this thesis, the author attempts to formulate a system for evaluation of code quality which is then used to grade a number of code patterns. These are ’pause()’, ’the combination of the reactivity of Vue and both custom properties and calc() of CSS’ and ’centralized handling of global state with Vuex.’ The origin of the patterns is the game Taloussankari by Talous ja nuoret TAT and 10Monkeys. The author claims almost full authorship of, and therefore familiarity with, the code for game.

The evaluation system is based on literature by, among others, McConnell (2004), Gamma et al. (1995), Brown et al. (1998), and the International Organization for Standardization (2011). The selection process of the code patterns is based mostly on the author’s familiarity with the code base.

The thesis does find that especially ’centralized handling of global state with Vuex’ is a very preferrable pattern and that the ’pause()’-pattern can be very problematic in certain use cases. It is found, however, that the evaluation system proposed coupled with its implementation in the thesis is severely lacking. Thus, the results of the evaluation cannot be considered reliable.

It is speculated, though, that the system could still be used with diagnosticity in mind, when a problem has been sensed in a code base on a general level, as a tool for structured critical thinking about code provided to the programmer.

Keywords:	JavaScript, Nuxt, Vue, code quality, web development, programming
Number of pages:	56
Language:	Swedish
Date of acceptance:	

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Online media
Tunnistenumero:	8676
Tekijä:	Martin Fellman
Työn nimi:	Menetelmä lähdekoodin laadun määrittelemiseksi – Analyysi Taloussankari Junior -peliä varten Nuxt-metakehyksessä toteutetuista koodirakennekuvioista
Työn ohjaaja (Arcada):	Mirko Ahonen
Toimeksiantaja:	

Tiivistelmä:
Tässä opinnäytteessä pyritään luomaan menetelmän lähdekoodin laadun määrittelyyn sekä käyttämään sitä määrittelemään joukon koodikuvioiden laatua. Nämä koodikuviot ovat ”pause()”, ”yhdistelmä Vuen reaktiivisuudesta ja CSS:n sekä räätälöidyistä ominaisuuksista että calc()-toiminnosta” ja ”keskitetty ohjelmatilan hallinta Vuex:n avulla”. Koodikuviot ovat peräisin Talous ja nuoret TAT:n ja 10Monkeysin pelistä Taloussankari Junior, jonka teknisestä toteutuksesta kirjoittaja oli miltei yksin vastuussa.

Menetelmä perustetaan m.m. McConnellin (2004), Gamman ym. (1995), Brownin ym. (1998) ja ISO:n (2011) kirjallisuuteen. Koodikuvioiden valinta perustuu kirjoittajan kokemukseen Taloussankari Junior -pelin koodista.

Työssä todetaan varsinkin ”keskitetyn ohjelmatilan hallinnan Vuex:n avulla” olevan menetelmän mukaan toimiva kuvio, kun puolestaan ”pause()”-kuvio tietyillä tavoilla käytettynä on hyvinkin ongelmallinen. Todetaan kuitenkin ettei menetelmä sovellu opinnäytetyön tapaan sovellettuna, joten tuloksia ei voida pitää luotettavina.

Ajatellaan tästä huolimatta että menetelmää pystyttäisiin todennäköisesti hyödyntämään koodin diagnostiikassa ohjelmoijan apukeinona ongelmien ymmärtämiseen ja ratkaisujen esittämiseen.

Avainsanat:	JavaScript, Nuxt, Vue, lähdekoodin laatu, verkkokehitys, ohjelmointi
Sivumäärä:	56
Kieli:	ruotsi
Hyväksymispäivämäärä:	

INNEHÅLL

Tabeller

Kodsnuttar

1 INTRODUKTION

1.1 Motiv för ämnesvalet

Omkring sommaren 2021 var skribenten genom sin arbetsplats utvecklare under skapandet av spelet *Taloussankari Junior (Talous ja Nuoret TAT och 10Monkeys.com Ltd 2021)* och hade relativt fria händer gällande spelets tekniska implementering. Han passade på att i spelet ta i bruk ett antal idéer han haft kring hur en del moderna JavaScript- och CSS-egenskaper inom ramverket Nuxt kunde användas.

Att ingående reflektera över och analysera de resulterande lösningarna har således varit ett intresse för skribenten ända sedan arbetet slutfördes, och står därmed som motiv för ämnesvalet i detta arbete.

1.2 Bakgrundsinformation

Taloussankari Junior (Talous ja Nuoret TAT och 10Monkeys.com Ltd 2021), som detta arbete behandlar, är ett spel som företaget skribenten arbetar för, *10Monkeys.com Ltd* (hädanefter *10Monkeys*), utvecklade under tiden juni-september 2021 på uppdrag av föreningen *Ekonomi och ungdom TAT* (10Monkeys.com Ltd 2021). Spelet är ett textbaserat inlärningsspel för barn i lågstadieåldern med målet att lära spelaren grunderna i ekonomiskt, hållbart och ekologiskt tänkande. Spelet är det andra spelet i serien *Taloussankari*, vars första spel, *Taloussankari* (2018), utvecklades under hösten 2018 och är riktat till 13–17-åringar (10Monkeys.com Ltd 2020; Talous ja Nuoret TAT och 10Monkeys.com Ltd 2018). Det spelet i sin tur går djupare in på att exempelvis planera sin egen ekonomi, tjäna pengar och investera.

1.3 Syfte med arbetet

Det yttersta syftet med detta arbete är att belysa hur väl några mönster möjliggjorda med kombinationen av Nuxt och modern JavaScript och CSS fungerar i praktiken och där-

med både bidra med idéer till hur dessa kan användas och höja en liten röst i diskussionen kring hur de moderna egenskaperna i JavaScript fungerar.

Att analysera koden är också utvecklande för skribenten; han tog flera risker i *Taloussankari Junior*, så reflektion i efterhand hjälper honom att förbättras som webbutvecklare.

Nästan lika viktigt som ovanstående blir systemet för att utvärdera kodkvalitet på en nivå högre än den som normalt erbjuds av till exempel verktyg för statisk kodanalys. Att ha ett generellt och relativt kort system för att åstadkomma det skulle vara väldigt nyttigt för alla programmerare, inte endast skribenten.

1.4 Frågeställningar och hypoteser

I detta arbete identifieras ett antal logiska mönster möjliggjorda av kombinationen av Nuxt och moderna JavaScript- och CSS-egenskaper som användes i *Taloussankari Junior*. Dessa utvärderas enligt ett av skribenten presenterat system för kortfattad evaluering av kodmönster.

Således är forskningsfrågorna i detta arbete följande:

1) Kan ett kortfattat system för utvärdering av kodmönster utformas, och hur väl fungerar det?

2) Hur utvärderas de utvalda mönstren enligt svaret på fråga 1?

Hypoteserna är följande:

1) Ett sådant system är möjligt, troligtvis med vissa brister.

2) Mönstren fungerar mestadels bra, med ett antal svagheter, och i de flesta fall objektivt att föredra framom att inte utnyttja dem.

1.5 Avgränsningar

Flera modeller för mjukvarukvalitet, närmare diskuterade i kapitel 2, lägger stor vikt på kodbasens testtäckning. Det är något som inte detaljerat diskuteras i detta arbete efter-

som det i ett tidigt skede i utvecklingen av *Taloussankari Junior* beslöts att projektet inte kommer att inkludera några enhetstest eller dylika automatiserade system för testande, utan att all kvalitetsgranskning sker manuellt.

Ett annat område som ofta betonas är programmets snabbhet i teknisk mening. Diskussioner kring detta avgränsas också från detta arbete; spelet är i grunden så enkelt och till sin genre sådant att maximering av dess prestanda inte är nödvändigt. Eventuella uppenbart långsamma lösningar rättades till i utvecklingsskedet.

En mer teknisk avgränsning är också att koden som används för stil i detta arbete är CSS, trots att *Taloussankari Junior* utnyttjar SCSS (se *CSS* och *SCSS* i avsnitt 1.6), förutom i de fall SCSS uttryckligen nämns. Orsaken till detta är att *SCSS* i de flesta fallen endast är en enkel utvidgning som gör arbetet lättare för programmeraren utan att direkt ge någon ytterligare funktionalitet. SCSS är dessutom en ”transpilerare” (eng. *transpiler*) till CSS, vilket betyder att all SCSS-kod programmatiskt omskrivs till CSS-kod före den sänds till användarens webbläsare, så all SCSS-kod kunde från början ha skrivits i SCSS. Att direkt behandla CSS gör också utsträckningen för ämnets relevans större.

1.6 Definitioner

Alfaversion: Den första versionen av en mjukvaruprodukt som testas formellt, t.ex. internt inom ett företag eller med en fokusgrupp. Alfaversionen förväntas ha buggar och behöver inte ha all planerad funktionalitet. (Wikipedia: ”Software release life cycle” 2022)

Asynkron (eng. asynchronous*):* Inom programmering: motsatsen till *synkron*; kod vars utföringsordning och/eller tidsbundenhet påverkas av externa faktorer, t.ex. så att den utförs vid en händelse eller på en skild tråd i processorn. (Costa 2021)

Bugg: Ett fel i ett datorprogram som gör att programmet på något sätt inte beter sig som förväntat. (Wikipedia: ”Software bug” 2022)

Buggfix: Processen att eliminera en bugg. (Wikipedia: ”Software bug” 2022)

CSS: Det kodspråk som används för att definiera (främst) den grafiska utformningen av innehåll på webben. Förkortning för *Cascading Style Sheets* (ung. *Nedåtflödande Stilmall*). (MDN 2022c)

Funktion: I JavaScript är en funktion ett kodblock som får något eller några värden som inmatning och returnerar något varde som utmatning, med eller utan övriga effekter på programmet. (MDN 2022e)

Gränssnitt (eng. interface*):* En anslutning mellan två enheter genom vilken de samverkar. (Wikipedia: ”Interface (computing)” 2022)

HTML: Det märkspråk som används för att strukturera och ge mening och sammanhang till innehåll på webben. Förkortning för *Hypertext Markup Language* (ung. *märkspråk för hypertext*). (MDN 2022f; Wikipedia: ”Märkspråk” 2019)

Klient (eng. client*):* Den begärande parten i en klient-server-struktur. Inom webbutveckling används termen ofta synonymt med ”webbläsare”. Se även *server*. (Wikipedia: ”Client-server model” 2022)

Metaramverk (eng. meta-framework*):* Ett ramverk kring ett ramverk. Se även *ramverk*.

Metod: En funktion som är kopplad till ett objekt och som således kan nå och/eller modifiera dess data. Se även objekt. (Wikipedia: ”Object-oriented programming” 2022)

Mjukvaruprototyp: En version av en mjukvaruprodukt som demonstrerar en eller flera av dess tilltänkta egenskaper för utvärdering ur ett eller flera perspektiv. Denna version behöver inte vara generellt fungerande och kan vara helt separat från den egentliga produkten som utvecklas. (Wikipedia: ”Software prototyping” 2021)

Nuxt: Ett metaramverk för Vue. Närmare information i avsnitt 2.2.

Objekt: Ett paradigm inom programmering enligt vilken kod abstraheras som ”objekt” som innehåller data (ofta kallade ”attribut”) och kod (ofta kallad ”metoder”). Se även *metod*. (Wikipedia: ”Object-oriented programming” 2022)

Omstrukturering (eng. refactoring*):* Att förbättra kvaliteten på programkod genom gradvisa förändringar för att t.ex. introducera nya logiska kopplingar eller ge klarare namn till entiteter i koden. (Wikipedia: ”Code refactoring” 2022)

Ramverk (eng. framework*):* En abstraktion kring en teknologi som underlättar användningen av teknologin för något specifikt ändamål. (Wikipedia: ”Software framework” 2022)

rem (måttenhet): Typsnittsstorleken på rotelementet på en webbsida. (MDN 2022b)

Sass: En förbehandlare (eng. *preprocessor*) för CSS som erbjuder diverse programmatiska hjälpmedel som underlättar programmerandet av CSS. Sass-kod omvandlas till CSS-kod före den används. Se även *CSS*. (Sass u.å. a)

SCSS: Förkortning av *Sassy CSS*. Ursprungligen en alternativ syntax för Sass, numera den förvalet. Jämfört med Sass påminner SCSS mer om CSS, men båda har exakt samma egenskaper. Se *Sass*. (Sass u.å. b)

Server (eng. server*):* Den levererande parten i en klient-server-struktur. Inom webbutveckling avser termen ofta en webbserver. Se även *klient*. (Wikipedia: ”Client-server model” 2022)

Synkron (eng. synchronous*):* Inom programmering: kod som utförs direkt i sekvens, utan inverkan utifrån på hur eller när den körs. Se även dess motsats, *asynkron*. (Costa 2021)

Syntax: Inom programmering: reglerna för hur kod i ett programmeringsspråk kan skrivas. (Wikipedia: ”Syntax (programming languages)” 2022)

Vue: Ett ramverk för JavaScript. Närmare information i avsnitt 2.2.

Webbapp: En webbsida med avancerad funktionalitet för vilken funktionaliteten, snarare än det informationsinnehåll som sidans skapare gett upphov till, är i fokus, likt traditionella appar som installeras på den apparat man använder. Typiska exempel är chatt-, videosamtals- och dokumentredigeringstjänster. (Wikipedia: ”Web application” 2022)

Webbsida: Ett dokument i HTML-format som från en server levereras till och visas i en webbläsare då man besöker en kompatibel webbadress, t.ex. https://exempel.com, https://annatexempel.com/rutt/till/resurs.html eller http://123.45.67.89. Webbsidor kan utformas (främst) grafiskt med programmeringsspråket CSS och ges olika typer

av dynamisk funktionalitet med programmeringsspråket JavaScript. (Wikipedia: ”Web page” 2022)

1.7 Strukturen

I kapitel 2 *Teoretisk referensram* behandlas den huvudsakliga teorin som behövs för detta arbete. En kort historia som placerar området som behandlas i historiskt perspektiv ges i avsnitt 2.1, de huvudsakliga teknologierna introduceras i avsnitt 2.2, och akademiskt material som berör ämnena kod- och mjukvarukvalitet presenteras i avsnitt 2.3.

I kapitel 3 *Metoder* presenteras grunderna för urval av material att analysera (avsnitt 3.1) samt metoden med vilken det analyseras (avsnitt 3.2).

I kapitel 4 *Resultatredovisning* presenteras i avsnitt 4.1 det material som utvalts för analys. I avsnitt 4.2 presenteras sedan analysen..

I avsnitt 5 Diskussion presenteras och diskuteras resultaten först, varefter styrkorna och svagheterna med arbetets material, metoder och resultat diskuteras på ett mer generellt plan.

2 TEORETISK REFERENSRAM

2.1 Från informationsteori till JavaScript-ramverk – en kort historia

Den teoretiska grunden för det som idag kallas ”internet” lades under 1920, 1930 och 1940-talen i och med forskning inom informationsteori utförd av bl.a. Nyqvist (Wikipedia: ”Harry Nyquist” 2022), Hartley (Wikipedia: ”Ralph Hartley” 2021) och Shannon (Wikipedia: ”Claude Shannon” 2022) samt utvecklingen av transistorn, vilken ursprungligen demonstrerades 1948 av (Bardeen och Brattain 1948) under ledning av Shockley. (Wikipedia: ”History of the Internet” 2022)

Den fysiska implementeringen av Internet fick sin början i och med *ARPANET*, utvecklat vid *Advanced Research Projects Agency* (ung. *myndigheten för avancerade forskningsprojekt*) vid USA:s försvarsdepartement (eng. *U.S. Department of Defense*) år 1969 (Wikipedia: ”ARPANET” 2022). *ARPANET* var ett av de första paketförmedlande nätverken med distribuerad kontroll och det första geografiskt brett utspridda sådant, och ett av de första nätverken att utnyttja den protokollsamling, kallad *DoD-modellen för internetarkitektur* (eng. *The DoD Internet Architecture Model*) (Cerf och Cain 1983), som senare utvecklades till internetprotokollen, eller *TCP/IP*. (Wikipedia: ”History of the Internet” 2022)

Webben (eng. *World Wide Web*) fick å sin sida sin början år 1989 i och med Berners-Lees förslag till och senare implementation av protokollet *HTTP*, märkspråket *HTML,* den första webbläsaren *WorldWideWeb,* den första webbservern *httpd* samt den första webbsidan. (Berners-Lee 1989, 1991; Wikipedia: ”World Wide Web” 2022)

Märkspråket HTML kompletterades inom några år av programmeringsspråken JavaScript och CSS. JavaScript utvecklades av Brendan Eich år 1995 för webbläsaren *Netscape Navigator* för att möjliggöra dynamiskt modifierat innehåll på webbsidor (speakingjs.com 2020). CSS föreslogs av Wium Lie (1994) och resulterade i en rekommendation av W3C upplagd av Wium Lie och Bos (1996) två år senare.

Redan Berners-Lee tänkte på det möjliga behovet av dynamiskt skapade webbsidor i sin implementering av *httpd:*

> Vi har också kod för en hypertextserver. Denna kan användas för att göra filer tillgängliga (likt anonym FTP men snabbare i och med att den bara använder en enda uppkoppling). **Den kan också modifieras så att den använder en hypertextadress och genererar ett virtuellt hypertextdokument från vilken annan data som helst – databas, föränderlig data osv.** Det handlar bara om att generera ren text eller SGML-kod (usch! men[sic] standard) enligt behov. Webbläsarna tolkar den sedan utan problem.

(Berners-Lee 1991, fritt översatt med tillagd emfas)

Detta lade grunden för de första ramverken för webben, och de första serverspråken lanserades år 1995, t.ex. PHP (Wikipedia: ”PHP” 2022) och Adobe ColdFusion (Wikipedia: ”Adobe ColdFusion” 2022). I slutet av 1990-talet fick sedan de första ramverken avsedda att sammanfoga webbläsarens och serverns funktionalitet fotfäste, varefter ramverken började splittras efter ändamål. (Wikipedia: ”Web framework” 2022)

En grov undergrupp av webbramverk efter denna splittring är ramverk för webbläsaren, eller användargränssnittsramverk. Ett av dessa är React.js, lanserat 2013 (Wikipedia: ”React (JavaScript library)” 2022), som enligt webbplatsen Stack Overflow (2021) är det mest populära ramverket för webbutveckling. År 2016 lanserades React-ramverket (”metaramverket”) Next.js, som erbjuder verktyg för att utvidga funktionaliteten hos React med t.ex. förrendering på servern (Wikipedia: ”Next.js” 2022).

Ett annat användargränssnittsramverk är Vue.js, lanserat år 2014 (Wikipedia: ”Vue.js” 2022), som enligt samma studie som ovan av Stack Overflow (Stack Overflow 2021) placeras på femte plats i popularitet. Vue.js å sin sida fick motsvarande metaramverksutvidgning som React.js fick av Next.js med Nuxt.js, också lanserat år 2016 och med motsvarande funktionalitet (Wikipedia: ”Nuxt.js” 2022). *Taloussankari Junior*, som detta arbete behandlar, är utvecklat med Nuxt.js.

2.2 JavaScript-ramverket Vue och metaramverket Nuxt

Nuxt är ett metaramverk för ramverket Vue. Vue beskrivs av Vue.js (u.å. c, fritt översatt från engelska) enligt följande:

> Vue [...] är ett JavaScript-ramverk avsett för att skapa användargränssnitt. Det bygger på vanliga HTML, CSS och JavaScript, och erbjuder en deklarativ och komponentbaserad programmeringsmodell som låter en utveckla såväl enkla som invecklade användargränssnitt effektivt.

Vue.js (u.å. c) lyfter fram två egenskaper som centrala för ramverket: deklarativ rendering, som låter en använda JavaScript-tillstånd i HTML, och reaktivitet, dvs. att tillståndsförändringar i JavaScript automatiskt upptäcks och förmedlas till webbsidesvyn.

Som metaramverk erbjuder Nuxt ett antal tillägg till och kloka förval i Vue som ytterligare underlättar utvecklingen med Vue, bl.a. ett automatiskt system för *routing* (ung. *ruttbestämning*), möjlighet att förrenderera sidor på servern och enklare komponentskapande. (Nuxt.js u.å.)

I detta arbete är några egenskaper hos Vue och Nuxt i särskild fokus. Dessa beskrivs till följande.

2.2.1 Layouter, sidor och komponenter i Nuxt och Vue

Vue baserar sig på komponenter, vilka låter innehåll, utformning och logik bli enkelt återanvända (Vue.js u.å. a). Nuxt erbjuder därtill ett system genom vilket sidor långt fungerar som komponenter, och dessutom ett layoutsystem med vilket en sida kan tilldelas en layout så att sidorna beter sig ungefär som komponenter i layouten, som i sig själv också är en komponent (Nuxt.js 2022b, 2022a; Vue.js u.å. b). En komponent läggs vanligtvis in i programmets hierarki med en icke-standard HTML-tagg – i kodsnutt 1 visas ett exempel på en komponenthierarki i Nuxt. Den hierarkin kunde ha filstruktur enligt kodsnutt 2.

Kodsnutt 1. Exempel på komponenthierarki i Nuxt

```
<min-layout>
  <layout-komponent-1 />
  <min-sida>
    <sidkomponent-1 />
    <sidkomponent-2>
      <komponent-i-komponent />
    </sidkomponent-2>
  </min-sida>
  <layout-komponent-2 />
</min-layout>
```

Kodsnutt 2. Exempel på filstruktur för en app i Nuxt

```
mitt-projekt/
  ├ ...
  │
  ├ components/
  │    ├ sidkomponent-2/
  │    │    └ KomponentIKomponent.vue
  │    │
  │    ├ Sidkomponent1.vue
  │    └ Sidkomponent2.vue
  │
  ├ layouts/
  │    └ MinLayout.vue
  │
  ├ pages/
  │    └ min-sida.vue
  │
  └ ...
```

2.2.2 Singulärfilskomponenter (eng. *Single file components*) som förval i Nuxt

Nuxt erbjuder singulärfilskomponenter, valbara i Vue, som förval. Singulärfilskomponenter låter en definiera sin HTML-mall (eng. *template*), JavaScript och CSS i en och samma fil som i kodsnutt 3, vilket underlättar arbetandet med komponenter.

Kodsnutt 3. Grundstrukturen för en singulärfilskomponent

```
<template>
  <!-- Komponentens HTML-mall här. -->
</template>

<script>
// Komponentens JavaScript här.
</script>

<style>
/* Komponentens CSS här. */
</style>
```

2.2.3 Reaktivitet i Vue

En av de mest framstående egenskaperna i Vue är den reaktiva modellen för data. I Vue uttrycker man inte explicit att man ändrar på en komponents tillstånd (till skillnad från t.ex. React.js (u.å.)), utan Vue implementerar ett system för att lyssna efter ändringar.

I kodsnutt 4 , som visar ett exempel på reaktiviteten i Vue, ökar värdet av numret `num` som visas med 1 varje gång användaren klickar på det, samtidigt som värdet som står efter ”Dubbelt:” ökar med två som ett värde kalkylerat baserat på `num` och `watch`. - `numTimesTwo()` kallas och skriver info om vad som skett till webbläsarens konsol. Det förutsätter att reaktivitetssystemet registrerat `num`, vilket kan åstadkommas på ett antal olika sätt; det vanligaste är att registrera reaktiva data i komponentens `data`-metod. Reaktiva data kan sedan användas och modifieras så gott som hur som helst, med förändringarna automatiskt reflekterade i webbvyn.

Kodsnutt 4. Exempel på reaktiviteten i Vue

```
<template>
  <div @click="incrementNumber">
    {{ num }}
  </div>
  <div>
    Dubbelt: {{ numTimesTwo }}
  </div>
</template>

<script>
export default {
  data () {
    return {
      num: 0
    }
  },
  computed: {
    numTimesTwo () {
      return (this.num * 2)
    }
  },
  methods: {
    incrementNumber () {
      this.num += 1
    }
  },
  watch: {
    numTimesTwo (newValue, oldValue) {
      console.log(
          'numTimesTwo uppdaterades reaktivt. ' +
          'Det nya respektive gamla värdet är',
          newValue,
          oldValue
      )
    }
  }
}
</script>
```

2.2.4 Global lägeshantering (eng. *Global state management*) med Vuex som förval i Nuxt

Nuxt (2022c) inkluderar lägeshanteringsramverket Vuex som förval. Vuex erbjuder global, strömlinjeformad lägeshantering som integreras i reaktivitetssystemet i Vue automatiskt via `this.$store`. Vuex har fem grundkoncept för datalagring och -åtkomst:

- Tillstånd lagras i ett objekt kallat `state`. Data däri är skrivskyddad – den kan läsas men inte direkt modifieras utifrån.
- En eller flera fattar- eller *getter*-metoder kan definieras för att utvidga den direkta åtkomstmodellen med icke-explicit data ur `state`. Ett typiskt exempel är att om `state` innehåller attributen `fornamn` och `efternamn` så kan en *getter* definieras för ett härlett attribut `helnamn = fornamn + efternamn`.
- Ändringar i tillståndet sker via mutationer, definierade i objektet `mutations`. Mutationer överlämnas till Vuex – `commit`. En mutation är alltid synkron.
- Handlingar – `actions` – kan definieras och avsändas – `dispatch` – för att sköta logik som överlämnar flera mutationer, möjligtvis asynkront.
- Data kan delas in i moduler.

(Vue.js 2022)

I Nuxt är Vuex implementerat så att varje fil i mappen `store` utgör en lagringsmodul (Nuxt.js 2022c). Grundmodulen för lagring är `store/index.js`, så ett enkelt spel där man samlar pengar och har någon mängd hälsa kunde ha filerna `store/money.js` för att hantera mängden pengar, `store/health.js` för att hantera mängden hälsa, och `store/index.js` för att t.ex. erbjuda ett centraliserat gränssnitt för att komma åt och hantera data.

2.3 Evaluering av kod- och mjukvarukvalitet

2.3.1 Kodkvalitet och programkvalitet som litteratur- och forskningsområden

Av de verk som behandlar kodkvalitet kan McConnells (2004), Martins (2009) och Hunts och Thomas (2000) nämnas. Diskuterade ämnen är till exempel hur variabler, datatyper och kontrollstrukturer kan användas för att hålla koden tydlig samt hur testande, buggfixande och omstrukturering kan utföras.

Kodkvalitet är ändå ett område som till en betydande del baserar sig på skribenternas erfarenhet, och kodkvalitet har också en stor estetisk aspekt; redan Knuth (1974) konstaterade detta då han diskuterade de artistiska aspekterna av programmering och programkod.

Mjukvarukvalitet är å sin sida ett ämne som åtnjutit en stor mängd forskning under flera årtionden i och med att mjukvarukvalitet är en av de faktorer som mest kan påverka ett projekts tidtabell och kostnader, som t.ex. Barbacci m.fl. (2000) påpekar. Det finns internationella standarder för mjukvarukvalitet, till exempel ISO 25010-standarden (Internationella standardiseringsorganisationen 2011), som för en mjukvaruprodukt ställer krav på bland annat dess användbarhet, prestanda, säkerhet och portabilitet.

2.3.2 Designmönster (eng. *design patterns*) och antimönster (eng. *anti-patterns*)

I stark koppling till frågan om kodkvalitet står designmönster inom mjukvaruutveckling. Dessa är allmänt vedertagna lösningar på ofta förekommande problem. De kan inte paketeras som färdig kod som kan kopieras in i ett projekt, utan beskriver snarare koncept som programmeraren kan applicera för att skapa lösningar vars kvalitet påvisats. Det har beskrivits väldigt många designmönster, vissa domänspecifika, och de är till största delen väldigt tekniska och abstrakta. (Wikipedia: ”Software design pattern” 2022)

Bland dem som haft mest inflytande på området är Gamma m.fl. (1995). De beskriver problem som förekommer i mjukvaruutveckling med exempel samt introducerar på ett strukturerat sätt ett stort antal designmönster som kan användas för att tackla dem.

Som en något sarkastisk kommentar till litteraturen om designmönster men ändå med allvarligt innehåll står Brown m.fl:s (1998) verk om antimönster. De å sin sida identifierar ett antal problematiska mönster, både tekniska och lednings-strukturella, som är allmänt förekommande och naturliga, vad de beror på och hur de kan åtgärdas.

2.3.3 System för utvärdering av kod- och mjukvaruproduktkvalitet

För utvärderingen av kodmönstren i *Taloussankari Junior* är skribenten i första hand intresserad av relativt kompakta, generella och holistiska modeller för att utvärdera kodkvalitet. En utmärkt sådan presenteras av McConnell (2004:464–65, fritt översatt med ändrad hierarkisk märkning):

- Underhållsmässighet: Hur enkelt det är att modifiera ett mjukvarusystem för att ändra på eller utöka dess funktionalitet, förbättra dess prestanda eller åtgärda felaktigheter hos det.
- Flexibilitet: I vilken mån det är möjligt att tillämpa ett system för användningsområden eller omgivningar utanför dess ursprungliga ändamål.
- Portabilitet: Hur enkelt det är att anpassa ett system för verksamhet i en annan omgivning än det ursprungligen planerats för.
- Återanvändbarhet: I vilken mån och hur enkelt delar av ett system kan användas i andra system.
- Läsbarhet: Hur enkelt det är att läsa och förstå ett systems källkod, särskilt på en detaljerat-beskedlig nivå (eng. *detailed-statement level*).
- Möjlighet att testas: I vilken grad ett system kan enhets- och systemtestas (eng. unit-test och system-test); i vilken grad det går att verifiera att ett system uppfyller de krav som ställts på det.
- Begriplighet: Hur enkelt det går att begripa ett system på en både system-organisatorisk och detalj-beskedlig nivå (eng. *system-organizational level* och *detailed-statement level*). Jämfört med läsbarhet handlar begriplighet om ett systems logiska struktur på en mer generell nivå.

McConnell (2004:464–65, fritt översatt, med ändrad hierarkisk märkning) presenterar också en motsvarande modell för mjukvarukvalitet. Hen syftar på dessa som externa kvalitetsattribut medan de tidigare var interna, och nämner:

- Korrekthet: I vilken grad ett system är felfritt till specifikation, utformning och implementering.
- Användbarhet: Hur enkelt det är för användare att begripa och använda ett system.
- Effektivitet: Att ett system använder möjligast lite systemresurser, medräknat minne och utföringstid.

- Pålitlighet: Förmågan för ett system att utföra de funktioner som krävs därav enligt de krav som ställs på det – att det sällan uppstår feltillstånd.

- Integritet: I vilken grad ett system förhindrar obehörig eller felaktig åtkomst till sina program och data. Integritet som koncept innefattar både begränsande av obehöriga användares åtkomster och att dataåtkomst sker på rätt sätt – med andra ord, att tabeller med parallella data modifieras parallellt, att fält för datum endast innehåller giltiga datum, och så vidare.

- Anpassningsförmåga: I vilken mån ett system i omodifierat tillstånd kan användas för andra tillämpningar eller i andra omgivningar än det ursprungligen planerats för.

- Exakthet: I vilken grad ett system till sin implementation är felfritt, särskilt vad gäller kvantitativ utskrift. Exakthet skiljer sig från korrekthet; det är ett mått på hur väl ett system utför det arbete det byggts för snarare än huruvida det byggts på rätt sätt.

- Robusthet: I vilken grad ett system fortsätter att fungera i fall av ogiltiga inmatningar eller utmanande förhållanden i sin omgivning.

ISO 25010-standarden (Internationella standardiseringsorganisationen 2011, fritt översatt, med ändrad hierarkisk märkning) presenterar en holistisk modell för mjukvarukvalitet med följande attribut, vart och ett med flera underattribut:

- Funktionell lämplighet: Detta attribut representerar i vilken grad en produkt eller ett system erbjuder funktioner som uppfyller dess ställda och implicerade krav då den eller det används i specificerade förhållanden.

- Prestandamässig effektivitet: Detta attribut representerar förhållandet mellan prestanda och använda resurser i specificerade förhållanden.

- Kompatibilitet: I vilken grad en produkt, ett system eller en komponent kan utbyta information med andra produkter, system eller komponenter och/eller utföra sina krävda funktioner då mjuk- eller hårdvaruomgivningen delas mellan dem.

- Användbarhet: I vilken grad en produkt eller ett system kan användas av avsedda användare för att uppnå specificerade mål effektivt, ändamålsenligt och tillfredsställande i en specificerad användningskontext.

- Pålitlighet: I vilken grad ett system, en produkt eller en komponent utför specificerade funktioner i specificerade förhållanden under en specificerad tidsperiod.

- Säkerhet: I vilken grad en produkt eller ett system skyddar information och data så att personer eller andra produkter eller system har den nivå av åtkomst till data som deras auktoriseringstyper och -nivåer föreskriver.

- Underhållsmässighet: Detta attribut beskriver hur effektivt och ändamålsenligt en produkt eller ett system kan modifieras för att förbättra den eller det, korrigera den eller det, eller anpassa den eller det till förändringar i omgivning och krav.

- Portabilitet: I vilken grad ett system, en produkt eller en komponent effektivt och ändamålsenligt kan överföras från en mjukvaru-, hårdvaru- eller annan operationell eller användningsomgivning till en annan.

Litteraturen om kodkvalitet är långt centrerad på att ge väldigt specifika råd om hur kod på bästa sätt kan skrivas (Barbacci m.fl. 2000; Martin 2009; McConnell 2004). Det är

inte ändamålsenligt att räkna upp dessa, men de kan användas på fallbasis för att evaluera någon eller några detaljer i källkod.

Det bör noteras att t.ex. McConnell (2004:465–66) konstaterar att de olika attributen kan ha mycket invecklade interaktioner. Då graden till vilken ett attribut uppfylls ökar kan ett eller flera andra attribut påverkas både positivt och negativt. Detta bör beaktas då eventuella alternativa lösningar till ett problem övervägs; som typiskt exempel kan ökandet av en kodkomponents underhållsmässighet, flexibilitet, portabilitet och/eller återanvändbarhet markant försämra dess läsbarhet och begriplighet medan dess möjlighet att testas till och med kan förbättras.

3 METODER

3.1 Val av kodmönster att utvärderas

Baserat på skribentens erfarenhet med och kännedom om koden i *Taloussankari Junior* väljs ett antal kodmönster ut för analys. Urvalet är i det stora hela subjektivt; kriterierna för urvalet är att kodmönstren skall vara icke-triviala samt representera någon intressant användning av Nuxt, Vue och/eller modern JavaScript och/eller CSS.

För varje mönster presenteras både en sammanfattande version i vilken grundidén med mönstret klart framkommer och den fullständiga versionen. I den fullständiga versionen visas för klarhets skull endast de delar som är relevanta för mönstret, och den ursprungliga stilkoden, skriven i SCSS, visas omskriven till CSS. Tillagda kodkommentarer kan förekomma.

3.2 System för utvärdering av kodmönstren

Med diskussionen i avsnitt 2.3 som bakgrund kommer följande system att användas för att utvärdera kodmönstren:

1. Mönstret utvärderas kvalitativt enligt de sju attributen för intern mjukvarukvalitet presenterade av McConnell (2004:464–65). För att underlätta explicit refererings till kriterierna numreras de enligt följande: *1 Underhållsmässighet*; *2 Flexibilitet*; *3 Portabilitet*; *4 Återanvändbarhet*; *5 Läsbarhet*; *6 Möjlighet att testas*; *7 Begriplighet*.

2. Det övervägs huruvida mönstret klart bryter mot de råd som presenteras av t.ex. Barbacci m.fl. (2000), Martin (2009) eller McConnell (2004) eller mot något av Gamma m.fl:s (1995) designmönster, eller följer något eller några av antimönstren presenterade av Brown m.fl. (1998), eller om det påverkar något av kvalitetsattributen för mjukvarukvalitet av Internationella standardiseringsorganisationen (2011) och McConnell (2004:464–65).

3. Förslag till hur mönstret kan förbättras ges, och någon nivå av omstrukturering presenteras vid behov.

4 RESULTATREDOVISNING

4.1 Presentation av de utvalda kodmönstren

4.1.1 `pause()`

I JavaScript hanteras tidsbundna avbrott i utförandet av kod traditionellt med återkallelser (eng. *callbacks*) och funktionen `setTimeout()` (MDN 2022g) som i kodsnutt 5. Som kodsnutt 6 däremot illustrerar blir det snabbt obehändigt att hantera invecklade serier av tidsbundna händelser då man som programmerare är tvungen att välja mellan två dåliga scenarion: flerstegskapsling och ”spagettikod”.

För att strömlinjeforma detta idiom skapade skribenten en hjälpfunktion, `pause()`, för vilken koden i sin helhet visas i kodsnutt 7. Genom att sedan utnyttja JavaScripts moderna *async/await*-struktur för asynkron programmering kan man i stället hantera de tre kallelserna i exemplet i kodsnutt 6 som en endimensionell kallelsekedja som i kodsnutt 8.

För att minimera mängden ändringar som behöver räknas upp enligt mönstret i kodsnutt 8 kombinerades stundvis `pause()`-funktionen med en `watch`-metod i Vue i likhet med exemplet i kodsnutt 9. Med det mönstret separeras följande två mekaniker:

- Text visas under två sekunder då knappen trycks på.
- Varje gång texten visas har den ändrats – den första gången ”Jag visas igen och igen”, den andra gången ”Jag visas igen och igen och igen” osv.

På detta sätt är det ytterst tydligt vad som händer när knappen trycks på, när texten visas, och då texten blir ändrad.

Kodsnutt 5. Trivialt exempel på metoden `setTimeout()` i JavaScript

```
// Kalla på funktionen `foo()` och en sekund senare
// på funktionen `bar()`.

foo()

setTimeout(() => {
  bar()
}, 1000)
```

Kodsnutt 6. Ett exempel på de problem som traditionellt uppstår vid tidsbundna pauser mellan kallelser i JavaScript

```
// Mål:
// - Kallelse 1
// - Paus på 1000ms
// - Kallelse 2
// - Paus på 500ms
// - Kallelse 3

// Flerstegskapsling
kallelse1()

setTimeout(() => {
  kallelse2()

  setTimeout(() => {

    kallelse3()

    // setTimeout( ... )
  }, 500)
}, 1000)

// Spagettikod
minKallelseKedja()

function minKallelseKedja () {
  kallelse1()

  setTimeout(minKallelseKedja2, 1000)
}

function minKallelseKedja2 () {
  kallelse2()

  setTimeout(minKallelseKedja3, 500)
}

function minKallelseKedja3 () {
  kallelse3()

  // setTimeout( ... )
}
```

Kodsnutt 7. Implementationen av funktionen `pause()` i sin helhet

```
function pause (ms) {
  return new Promise((resolve, reject) => {
    setTimeout(resolve, ms)
  })
}
```

Kodsnutt 8. Tidsbundna händelser i JavaScript med funktionen `pause()` och `async/await`-strukturen

```
minKallelseKedja()

async function minKallelseKedja () {

  kallelse1()

  await pause(1000)

  kallelse2()

  await pause(500)

  kallelse3()
}
```

Kodsnutt 9. Kombination av `pause()`-funktionen med en `watch`-metod i Vue

```
<template>
  <div>
    <button
        @click="onButtonClick"
    >
      Klicka på mig!
    </button>
    <p v-if="textShown">
      {{ text }}
    </p>
  </div>
</template>

<script>
export default {
  data () {
    return {
      textShown: false,
      text: 'Jag visas igen'
    }
  },
  methods: {
    async onButtonClick () {
      this.textShown = true
      await pause(2000)
      this.textShown = false
    },
    changeText () {
      this.text += ' och igen'
    }
  },
  watch: {
    textShown (isShown) {
      if (isShown) {
        this.changeText()
      }
    }
  }
}
</script>
```

4.1.2 Kombination av reaktiviteten i Vue och både anpassade egenskaper (eng. *custom properties*) och `calc()` i CSS

Traditionellt finns det två enkla metoder för att styra CSS-stilar med JavaScript: klasser och stilattribut. Vardera har sina begränsningar och problem:

- Styrning med klasser tillåter en inte godtyckligt applicera värden utan kräver att klasserna deklareras på förhand.

- Styrning med stilattribut låter en inte styra pseudoelement eller selektorer, och kan göra det otympligt att hantera kapslade element.

För att dynamiskt och precist kunna hantera logik, sidinnehåll och CSS-stilar använde skribenten ofta kombinationen av reaktiviteten i Vue och både anpassade egenskaper (MDN 2022d) och `calc()` (MDN 2022a) i CSS, som i kodsnutt 10. Då det exemplet renderas visas två kvadrater, en grå och en röd som går att trycka på med muspekaren. Den gråa kvadraten är placerad i fönstrets övre vänstra hörn och den röda så att dess övre vänstra hörn nuddar den gråa kvadratens nedre högra hörn. Varje gång den röda kvadraten trycks på ändras storleken på båda kvadraterna så att den gråa kvadratens sida upprepat antar värdena 1 rem, 2 rem, 3 rem, 4 rem och 5 rem, och den röda kvadraten är dubbelt så stor. Det nämnvärda i lösningen är att både storlekarna och applicerandet av dem sköts med CSS-kod genom en enda anpassad egenskap, `--num-value`, som modifieras i JavaScript, och att värdet på `--num-value` kan anta ett godtyckligt värde utan att behöva bindas till fördefinierade klasser eller liknande.

Kodsnutt 10. Kombination av reaktiviteten i Vue och anpassade egenskaper och `calc()` i CSS

```
<template>
  <div class="container">
    <div class="square passive" />
    <div
        class="square active"
        @click="num = (num + 1) % 5"
    >
      {{ num }}
    </div>
  </div>
</template>

<script>
export default {
  data () {
    return {
      num: 0
    }
  },
  watch: {
    num (value) {
      this.$el.style.setProperty(
          '--num-value',
          `${this.num}`
      )
    }
  }
}
</script>
<style scoped>
.container {
  --num-value: 0;
}
.container .square {
  position: absolute;
}
.container .square.passive {
  background-color: gray;
  top: 0;
  left: 0;
  width: calc(var(--num-value) * 1rem + 1rem);
  height: calc(var(--num-value) * 1rem + 1rem);
}
.container .square.active {
  background-color: red;
  top: calc(var(--num-value) * 1rem + 1rem);
  left: calc(var(--num-value) * 1rem + 1rem);
  width: calc(2 * var(--num-value) * 1rem + 1rem);
  height: calc(2 * var(--num-value) * 1rem + 1rem);
}
</style>
```

4.1.3 Central hantering av programtillstånd med Vuex

Med erfarenhet strävade skribenten efter att helt separera komponenters och programmets tillstånd i *Taloussankari Junior*. En komponents tillstånd kan som enkelt exempel

vara huruvida en meny är öppnad eller inte, och är i regel relevant enbart för komponenten och dess eventuella underkomponenter. Mängden tjänade pengar eller huruvida ett kapitel spelats är å sina sidor exempel på programtillstånd. Skillnaden är inte alltid helt klar, men som tumregel kan tillstånd som flera orelaterade komponenter behöver komma åt eller som bör vara mer eller mindre sparade under spelets gång anses vara programmets tillstånd.

I *Taloussankari Junior* utnyttjade skribenten Vuex väldigt liberalt för allt programtillstånd. Därtill lagras all data i möjligast små enheter, och åtkomst till data sker till stor del genom abstraherade gränssnitt i `store/index.js`. Ett exempel är hanterandet av pengar i *Taloussankari Junior*. I filen `store/money.js` definieras lagringsmönstret för pengar. För varje kapitel lagras mängden pengar som samlats ihop under dess gång. I `store/index.js` definieras sedan funktioner för att komma åt pengarna ur olika synvinklar – totala mängden pengar samt pengarna som kumulativt samlats ihop under spelets gång till och med något visst kapitel.

Varje lagringskomponent exponerar också metoder både för att samla ihop den data som bör sparas för att kunna återställa speltillståndet efter att spelet stängts och för att återställa motsvarande data då spelet startas på nytt. I grundkomponenten å sin sida finns metoder för att spara och ladda data, som automatiskt utnyttjar alla registrerade lagringskomponenters exporterings- och importeringsmetoder.

Implementeringen av ovanstående för mängden tjänade pengar per kapitel presenteras i bilaga 3.

4.2 Utvärdering av kodmönstren

4.2.1 `pause()`

Detta mönster visade sig väldigt kvalitativt på alla punkter så länge själva kedjan av turvisa kallelser och pauser bibehåller kontrollen över tidsbundenheten. Genom att definiera tidsintervallen som används med `pause()` som logiskt namngivna variabler var det dessutom enkelt att med teamet iterera sig fram till de optimala tidspauserna mellan kallelser. Således kan det konstateras att `pause()`-mönstret, utan `watch`, uppfyller alla

kriterier för intern kodkvalitet väl: det är atomiskt och litet, vilket innebär att kriterium *1 Underhållsmässighet* uppfylls väl; det är långt oberoende av omgivningen, vilket resulterar i att kriterierna *2 Flexibilitet*, *3 Portabilitet* och *4 Återanvändbarhet* uppfylls; det är ytterst kortfattat, saknar komplicerad logik som behöver förstås och har ett självbeskrivande namn, så kriterierna *5 Läsbarhet* och *7 Begriplighet* uppfylls; och kriterium *6 Möjlighet att testas* är långt beroende av omkringliggande kod.

Däremot visade det sig att den presenterade kombinationen av `pause()` och `watch` skapade stora problem både gällande hur lätt det gick att upprätthålla programmet och hur intuitiv koden var, och på grund av det omstrukturerade skribenten mycket av den kod där den kombinationen användes till att använda den så lite som möjligt. En del lämnades ändå i koden, och resultatet är inte tillfredsställande.

I bilaga 1 visas ett förkortat exempel från komponenten `ChannelMessage.vue` i *Talous-sankari Junior,* vilken sköter logiken för att i själva diskussionsvyn visa och dölja rätta knappar och pratbubblor i rätt ordning. I exemplet är det ytterst komplicerat att förstå vad som händer då spelaren trycker på knapparna i `section.channel-message .al-ternatives` då `v-else`-alternativet, `alternative-buttons`, visas:

- Metoden `onAlternativeClick()` kallas. Ur komponentens HTML-mall framgår inte att den kallas med det valda alternativet `alternative` som parameter.
- Logiken i `onAlternativeClick()` baseras på tre icke-triviala villkor: `isAl-ternativeContentTrivial(alternative)` (ung. är alternativets innehåll trivialt), `senderStaysTheSame` (ung. avsändaren förblir densamma) och `alterna-tiveCardIsExit` (ung. alternativkortet är ut). För enkelhetens skull antar jag hädanefter att alternativets innehåll inte är trivialt, vilket innebär att det i komponenten markeras att spelarens pratbubbla visas, diskussionspartnerns pratbubbla inte visas och alternativen inte visas, och att händelsen `alternative-click` signaleras (eng. *emit*).

Logiken ovan verkar vara någorlunda enkel. I praktiken är den ändå allt annat:

- Då det markeras att spelarens pratbubbla visas kallas `watch`-metoden för `playerSpeechBubbleShown`, som använder `pause()`-funktionen för att efter en

viss tid själv markera att spelarens pratbubbla inte visas, vilket *igen* kallar på den samma `watch`-metoden, vilket den gången resulterar i att det markeras att diskussionspartnerns pratbubbla visas.

- Då diskussionspartnerns pratbubbla visas används sedan en `watch`-metod för att efter en väntan definierad med `pause()`-funktionen markera att alternativen visas - vilket explicit i `onAlternativeClick()` markerades precis tvärtom.
- Då händelsen `alternative-click` signaleras kallas metoden `onAlternativeClick()` *i en annan komponent*. Den metoden ändrar efter komplicerad logik det aktuella kortet, `this.card`, i `ChannelMessage` (förutsatt att det nya kortet tillhör samma kanal som kortet tidigare).
- Det finns en `watch`-metod för card som inte har några eskalerande bieffekter. Men det finns *också* `watch`-metoder för `computed`-variablerna `senderId` och `visibleCharacters` som båda beror på kortet. De ändras inte varje gång kortet ändras, utan det finns en egen logik för huruvida de ändras eller inte och således för huruvida `watch`-metoderna för dem kallas eller inte.
- Till råga på allt ändrar `watch`-metoden för `visibleCharacters` på huruvida både spelarens och diskussionspartnerns pratbubblor visas med logik som inkluderar `pause()`-funktionen, och för den informationen konstaterades det redan att det finns `watch`-metoder som då igen kallas.

Med logiken utskriven är det uppenbart att det redan är svårt att förstå hur koden fungerar i sin nuvarande form. Det är ändå ingenting jämfört med vad utvecklaren behöver utstå för att ändra på den. Exempel på sådana fullständigt naturliga ändringar är att ändra på i vilken ordning händelserna sker eller att utvidga funktionaliteten.

Orsaken till logiken och lösningarna i exemplet ovan är att komponenten `ChannelMessage.vue` började som en väldigt liten prototyp som sedan utvecklades med respekt för den etablerade logiken. Lösningen med `pause()` och `watch`-metoder var fullständigt överskådlig då den inte var längre än några tiotals rader lång, men i takt med att den utvidgades och skulle omfatta allt flera situationer blev lösningen allt mindre intuitiv.

Det hade varit mer överskådligt att sköta komponentens logik utgående från en eller några centrala metoder med uppgift att delegera logiken strikt nedåt i hierarkin. En sådan skulle åtminstone behövas i det aktiva spelets sidkomponent, `channel/_id.vue`, och sedan kunde varje spelvykomponent, t.ex. `ChannelMessage.vue`, ha sin egen.

Det är alltså uppenbart att kombinationen av `pause()` med `watch` klarar sig väldigt svagt vad gäller alla kriterier förutom möjligtvis *4 Återanvändbarhet* och *5 Läsbarhet*, och det avslöjar orsaken till att skribenten använde mönstret: det ser elegant och kortfattat ut.

Mönstret är ett typiskt exempel på antimönstret *Spagettikod* (Brown m.fl. 1998:65–72) och har även drag av antimönstret *Lavaflöde* (Brown m.fl. 1998:49–54). Som (Brown m.fl. 1998) påpekar är lösningen till båda i korthet att omstrukturera och att prioritera arkitektur under processen att ta mjukvaran från prototyp till produktion.

4.2.2 Kombination av reaktiviteten i Vue och både anpassade egenskaper (eng. c*ustom properties*) och `calc()` i CSS

Den mest framstående användningen av detta mönster är i komponenten `Speech-Bubble.vue`, i logiken för att ändra storleken på spelets pratbubblor och på texten i dem baserat på textens längd. Funktionaliteten i fråga introducerades relativt sent i utvecklingsprocessen, och logiken för att generera pratbubblorna var redan väldigt invecklad. Ett kraftigt modifierat utdrag ur komponenten presenteras i bilaga 2.

Lösningen visade sig väldigt kraftig och underlättar arbetet med komplicerad stillogik storligen. Med lösningen är det möjligt att så gott som helt separera programlogik från stillogik, vilket gör programlogiken mer lättläst och -förstådd. Genom att sedan i stillogiken räkna ut flera anpassade egenskaper baserat på de egenskaper som levereras till den går det att hålla även stillogiken lättläst och uteslutande styrd från den avdelning av koden man förväntar sig hitta den i – från `<style>`-taggen. Denna lösning uppfyller därmed kriterierna *5 Läsbarhet* och *7 Begriplighet* utmärkt.

Mönstret håller även gränssnittet mellan programlogik och stillogik väldigt liten, vilket bidrar till att uppfylla kriterierna *1 Underhållsmässighet*, *2 Flexibilitet*, *3 Portabilitet* och *4 Återanvändbarhet*.

Att testa stilkod kan vara anmärkningsvärt svårt – CSS är i grunden avsett för enkla, statiska stildefinitioner, och de testramverk som existerar är i det stora hela avsedda för att t.ex. bekräfta att den renderade sidan och renderade komponenter motsvarar referensbilder (Burgmer 2022; Garris 2022; Shore 2022; Testim 2021). Således kan det konstateras att kriterium *6 Möjlighet att testas* inte uppfylls, särskilt jämfört med att sköta uträknandet i JavaScript.

Ytterligare *2 Flexibilitet*, *4 Återanvändbarhet*, *5 Läsbarhet* och *7 Begriplighet* fås i mönstrets ursprungliga implementering med SCSS. Med funktioner och mixins i SCSS är det möjligt att ytterligare skapa återanvändbara, generella och modulära kodenheter som fritt kan appliceras annanstans för andra ändamål och som gör koden ytterligare begripligare och mer läsbar. Med dem kan till och med kriterium *6 Möjlighet att testas* tänkas uppfyllas aningen bättre.

I Vue version 3.2 underlättas detta mönster vidare. Vue.js (2021) erkänner styrkan i kombinationen av reaktivitet och CSS i och med att de introducerar CSS-funktionen `v-bind()` i enfilskomponenter, vilket möjliggör användning av variabler i JavaScript direkt i CSS utan att behöva utföra mellansteget att binda dem till en anpassad egenskap.

Ovanstående tillägg kommer ändå inte utan möjliga nackdelar. Det gör till exempel att gränssnittet mellan programlogik och stillogik blir avsevärt bredare, vilket kan inverka negativt på kodens *1 Underhållsmässighet*, *2 Flexibilitet* och *3 Portabilitet*. Det är relativt lätt åtgärdat med systematik och disciplin; ett alternativ är att uteslutande hålla sig till en explicit definierad gränssnittsklass som i exemplet i kodsnutt 11.

Kodsnutt 11. Gränssnittsklass för `v-bind` i Vue

```
<script>
export default {
  // ...
  computed: {
    // ...
```

```
    css () {
      return {
        someVariable: this.someVariable,
        otherVariable: this.otherVariable
        // ...
      }
    }
    // ...
  }
  // ...
}
</script>

<style scoped>
/* ... */
.some-class {
  height: v-bind(css.someVariable)rem;
  width: calc(2rem * v-bind(css.otherVariable));
}
/* ... */
</style>
```

4.2.3 Central hantering av programtillstånd med Vuex

Detta mönster är i grund och botten det av mönstren som till störst utsträckning baserar sig på explicit avsedd användning av en egenskap i Nuxt: dess implementering av Vuex.

Så som mönstret utnyttjades i *Taloussankari Junior* är mönstret väldigt användbart. Den främsta negativa sidan med mönstret, inte heller den fatal, är att åtkomst till särskilt underkomponenter i Vuex har en något långrandig syntax. För att till exempel avsända handlingen `resetSomething` i modulen `something` med värdet `{ value: 0 }` är beteckningen `this.$store.dispatch('something/resetSomething', { value: 0 })`. Handlingen måste också implementeras i Vuex-lagringskomponenten i fråga vilket ökar spridningen av logik i programmet, så det kan konstateras att kodens *5 Läsbarhet* lider något.

Implementeringen av Vuex i Nuxt erbjuder inte heller något egentlig vägledning gällande *hur* eventuella omnämnda modulgränssnitt implementeras i grundmodulen. Det kan i värsta fall leda till att eventuella modulgränssnitts *1 Underhållsmässighet* och/eller *7 Begriplighet* lider – men det är endast en teoretisk risk till det och om det händer är det lika mycket på programmerarens ansvar.

I regel påstår skribenten ändå att användningen av Vuex är så gott som universellt fördelaktig – tack vare det var t.ex. funktionaliteten för att spara spelets tillstånd så gott som trivial att implementera och särskilt att modifiera och utöka. Funktionaliteten för att spara och ladda spelet var bland de första som implementerades och krävde inga större omskrivningar under utvecklingens gång; den utvidgades endast för att ta i beaktande nya data.

Under utvecklingens gång testades också flera olika modeller för hur mängden pengar visas och hanteras. Skribentens tidiga beslut att spara mängden tjänade pengar per kapitel och erbjuda *getters* för att komma åt olika kombinationer av penningvärden gjorde den processen snabb och smärtfri – endast nya enkla *getter*-metoder behövde implementeras.

På basis av ovanstående diskussion anser skribenten att alla kriterier 1–7 uppfylls för användningen av Vuex.

5 DISKUSSION

5.1 Sammandrag och diskussion av resultaten

Ett sammandrag av resultaten av analysen i kapitel 4.2.1–4.2.3 presenteras i tabell 1. ”+” innebär att skribenten anser att mönstret tydligt uppfylls, ”–” att han inte anser att det gör det. ”0” betyder att det är oklart eller beroende på något utanför mönstret som specificerat.

Tabell 1. Sammandrag av resultaten från analysen av kodkvalitet

Mönster	1	2	3	4	5	6	7
1. `pause()`	+	+	+	+	+	+	+
2. `pause()` *med* `watch`	–	–	–	0	0	–	–
3. Reaktivitet, anpassade egenskaper, `calc()`	+	+	+	+	+	–	+
4. Vuex	+	+	+	+	+	+	+

Som tabell 1 indikerar är tre av mönstren, 1, 2 och 4, kvalitativa, med mönster 3 det enda med ett minus för *6 Möjlighet att testas.* Detta motsvarar i stort skribentens erfarenhet av att arbeta med mönstren, men skillnaden i utvärdering mellan mönster 1 och 2 indikerar redan problem med utvärderingssystemet – mer om det nedan.

Mönster 2 är så gott som universellt negativt vilket, igen, reflekterar skribentens erfarenhet med att arbeta med mönstret.

Gällande själva utvärderingssystemet lämnar det mycket att önska. Det uppfyller kriterierna att vara kortfattat och generellt, men samtidigt är det enligt applikationen i detta arbete så gott som meningslöst. Då t.ex. mönster 1 och 2, av vilka mönster 2 egentligen är en undergrupp av mönster 1, får så polariserade resultat är det uppenbart att systemet inte säger mycket om mönstrens egentliga meriter.

Mer specifikt tar systemet så som det används i detta arbete knappt alls i beaktande de möjliga problem som kan uppstå i och med ett mönster fastän mönstret i sin grundläggande användning kan fungera bra. Systemet är också känsligt för utvecklarens erfarenhet, smak och prioriteter; en annan programmerare hade gott kunnat utvärdera mönstren inom dessa ramar mycket annorlunda än skribenten.

5.2 Sammanfattning

Härmed har skribenten som planerat demonstrerat ett system för utvärdering av kodkvalitet och utvärderat några utvalda mönster med det.

Som det konstaterades i avsnitt 5.1 är utvärderingssystemet inte lämpat för det ändamål som det i detta arbete används för. Baserat på den diskussionen verkar det som om det trots allt för programmeraren själv antagligen är mer produktivt att vara bekant med de mer detaljerade men mycket mindre universella rekommendationerna för god kodkvalitet som skribenterna inom ämnet oftast koncentrerar sina verk på. Detta är knappast överraskande; litteraturen skulle sannolikt se markant annorlunda ut om det var enkelt att skapa generella, korta mätare för kodkvalitet som kan användas på detta sätt: av en ensam programmerare, för kodsnuttar utan alternativa formuleringar.

Använt på andra sätt kan systemet ändå tänkas kunna fungera för att få mer meningsfulla resultat. Några alternativ är följande:

- Be en betydande mängd programmerare utvärdera mönstren, gärna så att bakgrundsinformation gällande specialiseringsområden och erfarenhet också kartlades. Det skulle ändå inte vara effektivt för denna typ av studie, i vilken koden inte är i behov av förbättring, då mängden tid som kollektivt skulle gå åt till det är väldigt stor. För att å andra sidan utveckla ett system som testas med denna metod skulle det vara viktigt att betydligt mer rigoröst planera systemet för utvärdering, vilket inte var möjligt enligt utsträckningen för detta arbete.

- Planera redan i utvecklingsskedet för en studie med mönstret genom att först göra en implementering utan de mönster som analyseras och sedan omorganisera koden med mönstren, för att sedan jämföra implementeringarna. På det sättet kunde vart och ett av utvärderingskriterierna baseras på kodsnuttarna isolerade

från sammanhanget. Det skulle ändå inte fungera för denna typ av tillbakablick; att göra en rättvis ”baklänges omorganisering” är svårt och subjektivt laddat.

Med utvärderingssystemet på så bräcklig grund är nyttan av att ge några generella utlåtanden om kodmönstrens kvalitet baserat på det även liten; skribenten hade format sin åsikt om kodmönstren på förhand, och systemet lät honom snarast formalisera sina åsikter.

Skribenten anser ändå inte att systemet saknar meriter. Det som systemet verkar kunna användas för, eller åtminstone lätt anpassas till, är som strömlinjeformat analysverktyg för att identifiera styrkor och svagheter i kodmönster av varierande utsträckning och modifiera dem baserat på det. Om ett beslut att något behöver förbättras fattats och en en lösning håller på att utvecklas kan systemet användas för att identifiera styrkorna med det samt vilka de är, för att låta programmerarna koncentrera sig på att rätta till de svagheter som återstår. Om inget problem ännu arbetas på kunde systemet i stället användas för att upptäcka de mest kritiska svagheterna i kodkvalitet för att få en startpunkt för förbättrandet.

I och med detta arbete upplever skribenten därmed att han fått värdefull insikt gällande hur kodkvalitet kan analyseras, motiveras, kritiseras och diskuteras. Under arbetets gång har han dessutom bekantat sig med ett flertal skribenter i ämnet, och han känner att han genom arbetet nått det viktigaste personliga målet: han har blivit en litet bättre och mycket mer analytiskt kapabel programutvecklare.

REFERENSER

10Monkeys.com Ltd. 2020. ”Money Master by 10Monkeys Digital”. Hämtad 06 maj 2022 (https://www.10monkeysdigital.com/en/work/taloussankari).

10Monkeys.com Ltd. 2021. ”Taloussankari Junior by 10Monkeys Digital”. *10Monkeys Digital*. Hämtad 06 maj 2022 (https://www.10monkeysdigital.com/en/work/taloussankari-junior).

Barbacci, Mario R., Robert J. Ellison, Charles B. Weinstock, och William G. Wood. 2000. *Quality Attribute Workshop Participants Handbook:* Fort Belvoir, VA: Defense Technical Information Center. doi: 10.21236/ADA455616.

Bardeen, J., och W. H. Brattain. 1948. ”The Transistor, A Semi-Conductor Triode”. *Physical Review* 74(2):230–31. doi: 10.1103/PhysRev.74.230.

Berners-Lee, Tim. 1989. ”Information Management: A Proposal”. Hämtad 06 maj 2022 (https://www.w3.org/History/1989/proposal.html).

Berners-Lee, Tim. 1991. ”Re: Qualifiers on Hypertext links...” Hämtad 06 maj 2022 (https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt).

Brown, William J., Raphael C. Malveau, Hays W. McCormick III, och Thomas J. Mowbray, red. 1998. *AntiPatterns: refactoring software, architectures, and projects in crisis*. New York: Wiley.

Burgmer, Christoph. 2022. ”CSS Critic”. Hämtad 06 maj 2022 (https://github.com/cburgmer/csscritic).

Cerf, Vinton G., och Edward Cain. 1983. ”The DoD Internet Architecture Model”. *Computer Networks (1976)* 7(5):307–18. doi: 10.1016/0376-5075(83)90042-9.

Costa, Ricardo. 2021. ”Asynchronous vs. Synchronous Programming: When to Use What”. Hämtad 08 maj 2022 (https://www.outsystems.com/blog/posts/asynchronous-vs-synchronous-programming/).

Gamma, Erich, Richard Helm, Ralph Johnson, och John Vlissides, red. 1995. *Design patterns: elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley.

Garris. 2022. ”BackstopJS”. Hämtad 06 maj 2022 (https://github.com/garris/BackstopJS).

Hunt, Andrew, och David Thomas. 2000. *The Pragmatic Programmer: From Journeyman to Master*. Reading, Mass: Addison-Wesley.

Internationella standardiseringsorganisationen. 2011. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO 25010:2011.

Knuth, Donald. 1974. ”Knuth: Computer Programming as an Art”. Hämtad 06 maj 2022 (http://www.paulgraham.com/knuth.html).

Martin, Robert C., red. 2009. *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall.

McConnell, Steve. 2004. *Code Complete*. 2nd ed. Redmond, Wash: Microsoft Press.

MDN. 2022a. ”Calc()”. *Calc()*. Hämtad 06 maj 2022 (https://developer.mozilla.org/en-US/docs/Web/CSS/calc).

MDN. 2022c. ”CSS: Cascading Style Sheets”. Hämtad 06 maj 2022 (https://developer.mozilla.org/en-US/docs/Web/CSS).

MDN. 2022b. ”CSS Values and Units”. Hämtad 08 maj 2022 (https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units).

MDN. 2022d. ”Custom Properties (--*): CSS Variables”. Hämtad 06 maj 2022 (https://developer.mozilla.org/en-US/docs/Web/CSS/--*).

MDN. 2022e. ”Functions”. Hämtad 08 maj 2022 (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions).

MDN. 2022f. ”HTML: HyperText Markup Language”. Hämtad 06 maj 2022 (https://developer.mozilla.org/en-US/docs/Web/HTML).

MDN. 2022g. ”SetTimeout()”. Hämtad 06 maj 2022 (https://developer.mozilla.org/en-US/docs/Web/API/setTimeout).

Nuxt.js. 2022a. ”Layouts Directory”. Hämtad 06 maj 2022 (https://nuxtjs.org/docs/directory-structure/layouts).

Nuxt.js. 2022b. ”Pages Directory”. *Nuxt*. Hämtad 06 maj 2022 (https://nuxtjs.org/docs/directory-structure/pages).

Nuxt.js. 2022c. ”Store Directory”. Hämtad 06 maj 2022 (https://nuxtjs.org/docs/directory-structure/store).

Nuxt.js. u.å. ”The Intuitive Vue Framework”. *Nuxt*. Hämtad 06 maj 2022 (https://nuxtjs.org/).

React.js. u.å. ”State and Lifecycle”. Hämtad 06 maj 2022 (https://reactjs.org/docs/state-and-lifecycle.html).

Sass. u.å. a. ”Sass Basics”. Hämtad 06 maj 2022 (https://sass-lang.com/guide).

Sass. u.å. b. ”Syntax”. Hämtad 06 maj 2022 (https://sass-lang.com/documentation/syntax).

Shore, James. 2022. ”Quixote”. Hämtad 06 maj 2022 (https://github.com/jamesshore/quixote).

speakingjs.com. 2020. ”Chapter 4. How JavaScript Was Created”. Hämtad 06 maj 2022 (https://web.archive.org/web/20200227184037/https://speakingjs.com/es5/ch04.html).

Stack Overflow. 2021. ”Stack Overflow Developer Survey 2021”. *Stack Overflow*. Hämtad 08 maj 2022 (https://insights.stackoverflow.com/survey/2021/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2021).

Talous ja Nuoret TAT och 10Monkeys.com Ltd. 2018. ”Taloussankari”. Hämtad 06 maj 2022 (https://taloussankari.com/).

Talous ja Nuoret TAT och 10Monkeys.com Ltd. 2021. ”Taloussankari Junior”. *Taloussankari Junior*. Hämtad 06 maj 2022 (https://www.taloussankarijunior.fi/?launchTime=1651824241313).

Testim. 2021. ”A Deep Dive Into CSS Testing: A Developer’s Complete Guide”. *AI-Driven E2E Automation with Code-like Flexibility for Your Most Resilient Tests*. Hämtad 06 maj 2022 (https://www.testim.io/blog/css-testing/).

Vue.js. 2021. ”Vue 3.2 Released!” Hämtad 06 maj 2022 (https://blog.vuejs.org/posts/vue-3.2.html).

Vue.js. 2022. ”What Is Vuex?” Hämtad 06 maj 2022 (https://vuex.vuejs.org/).

Vue.js. u.å. a. ”Components Basics”. Hämtad 06 maj 2022 (https://vuejs.org/guide/essentials/component-basics.html).

Vue.js. u.å. b. ”Components Directory”. *Nuxt*. Hämtad 06 maj 2022 (https://nuxtjs.org/docs/directory-structure/components/).

Vue.js. u.å. c. ”Introduction”. Hämtad 06 maj 2022 (https://vuejs.org/guide/introduction.html#what-is-vue).

Wikipedia: ”Adobe ColdFusion”. 2022. ”Adobe ColdFusion”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Adobe_ColdFusion&oldid=1079113973).

Wikipedia: ”ARPANET”. 2022. ”ARPANET”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=ARPANET&oldid=1085685047).

Wikipedia: ”Claude Shannon”. 2022. ”Claude Shannon”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Claude_Shannon&oldid=1083585325).

Wikipedia: ”Client-server model”. 2022. ”Client–Server Model”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=1079944730).

Wikipedia: ”Code refactoring”. 2022. ”Code Refactoring”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Code_refactoring&oldid=1086160643).

Wikipedia: ”Harry Nyquist”. 2022. ”Harry Nyquist”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Harry_Nyquist&oldid=1080204431).

Wikipedia: ”History of the Internet”. 2022. ”History of the Internet”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=History_of_the_Internet&oldid=1084941719).

Wikipedia: ”Interface (computing)”. 2022. ”Interface (Computing)”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Interface_(computing)&oldid=1072960630).

Wikipedia: ”Märkspråk”. 2019. ”Märkspråk”. *Wikipedia*. Hämtad 06 maj 2022 (https://sv.wikipedia.org/w/index.php?title=M%C3%A4rkspr%C3%A5k&oldid=46425912).

Wikipedia: ”Next.js”. 2022. ”Next.Js”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Next.js&oldid=1083713520).

Wikipedia: ”Nuxt.js”. 2022. ”Nuxt.Js”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Nuxt.js&oldid=1081143773).

Wikipedia: ”Object-oriented programming”. 2022. ”Object-Oriented Programming”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Object-oriented_programming&oldid=1085269845).

Wikipedia: ”PHP”. 2022. ”PHP”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=PHP&oldid=1086223937).

Wikipedia: ”Ralph Hartley”. 2021. ”Ralph Hartley”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Ralph_Hartley&oldid=1036157277).

Wikipedia: ”React (JavaScript library)”. 2022. ”React (JavaScript Library)”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=1085724690).

Wikipedia: ”Software bug”. 2022. ”Software Bug”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Software_bug&oldid=1068692494).

Wikipedia: ”Software design pattern”. 2022. ”Software Design Pattern”. *Wikipedia*. Hämtad 09 maj 2022 (https://en.wikipedia.org/w/index.php?title=Software_design_pattern&oldid=1086326856).

Wikipedia: ”Software framework”. 2022. ”Software Framework”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Software_framework&oldid=1068590172).

Wikipedia: ”Software prototyping”. 2021. ”Software Prototyping”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Software_prototyping&oldid=1061095107).

Wikipedia: ”Software release life cycle”. 2022. ”Software Release Life Cycle”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Software_release_life_cycle&oldid=1068238552).

Wikipedia: ”Syntax (programming languages)”. 2022. ”Syntax (Programming Languages)”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Syntax_(programming_languages)&oldid=1079968024).

Wikipedia: ”Vue.js”. 2022. ”Vue.Js”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Vue.js&oldid=1083614150).

Wikipedia: ”Web application”. 2022. ”Web Application”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Web_application&oldid=1084972116).

Wikipedia: ”Web framework”. 2022. ”Web Framework”. *Wikipedia*. Hämtad 08 maj 2022 (https://en.wikipedia.org/w/index.php?title=Web_framework&oldid=1085388396).

Wikipedia: ”Web page”. 2022. ”Web Page”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=Web_page&oldid=1084829025).

Wikipedia: ”World Wide Web”. 2022. ”World Wide Web”. *Wikipedia*. Hämtad 06 maj 2022 (https://en.wikipedia.org/w/index.php?title=World_Wide_Web&oldid=1086450451).

Wium Lie, Håkon. 1994. ”Cascading HTML Style Sheets -- A Proposal”. Hämtad 06 maj 2022 (https://www.w3.org/People/howcome/p/cascade.html).

Wium Lie, Håkon, och Bert Bos. 1996. ”Cascading Style Sheets, level 1”. Hämtad 06 maj 2022 (https://www.w3.org/TR/1999/REC-CSS1-19990111).

BILAGOR

Bilaga 1: Exempel på kombinationen av `pause()` och `watch`

```
<template>
  <section
      class="channel-message"
  >
    <!-- ... -->
    <div class="channel-message-container">
      <message-view
          :visible-characters="visibleCharacters"
          :card="card"
          :card-text-array="cardTextArray"
          :card-text-index="cardTextIndex"
          :player-speech-bubble-shown="playerSpeechBubbleShown"
          :character-speech-bubble-shown="characterSpeechBubbleShown"
          :previous-chosen-alternative="previousChosenAlternative"
          :tutorial-in-progress="tutorialInProgress"
      />
    </div>
    <!-- ... -->
    <div class="alternatives">
      <template v-if="alternativesShown || tutorialInProgress">
        <template v-if="tutorialInProgress">
          <message-button
              class="double-bubble-button"
              :text="threeDots"
              @button-click="progressTutorial()"
          />
        </template>
        <template v-else-if="(cardTextIndex < cardTextMaxIndex)">
          <message-button
              class="double-bubble-button"
              :text="threeDots"
              @button-click="progressDoubleBubble()"
          />
        </template>
        <template v-else>
          <alternative-buttons
              :alternatives="alternatives"
              :channel="'message'"
              @alternative-click="onAlternativeClick"
          />
        </template>
      </template>
    </div>
  </section>
</template>
```

```
<script>
// import ...
export default {
  // components: { ... }
  props: {
    card: {
      type: Object,
      default: null
    },
    alternatives: {
      type: Array,
      default () { return [null, null] }
    },
    previousChosenAlternative: {
      type: Object,
      default: null
    },
    visibleCharacters: {
      type: Array,
      default () { return [] }
    },
    chapterId: {
      type: String,
      default () { return null }
    },
    tutorialInProgress: {
      type: Boolean,
      default: false
    }
  },
  data () {
    return {
      characterSpeechBubbleShown: false,
      playerSpeechBubbleShown: false,
      alternativesShown: false,
      cardTextIndex: 0,
      content
    }
  },
  computed: {
    cardTextArray () {
      return (this.cardRawText?.split('---') || [])
    },
    cardTextMaxIndex () {
      return (this.cardTextArray.length - 1)
    },
    cardRawText () {
      return this.card?.text
    },
    threeDots () {
      return threeDots
    },
    cardOfAlternative () {
      return alternative =>
          cardOfAlternative(this.$store, alternative)
    },
    isCharacterVisible () {
      return characterId =>
          this.$store.state
              .characterVisibility[this.chapterId][characterId]
    },
```

```
  characterNameToId () {
    return characterName => characterNameToId(characterName)
  },
  alternativeToFullText () {
    return alternative => alternativeToFullText(alternative)
  },
  senderId () {
    return this.characterNameToId(this.card.sender)
  }
},
watch: {
  async visibleCharacters () {
    this.characterSpeechBubbleShown = false
    this.alternativesShown = false
    this.playerSpeechBubbleShown = false
    await pause(timings.beforeInitialCharacterSpeechBubble)
    this.characterSpeechBubbleShown = true
  },
  async characterSpeechBubbleShown (isShown) {
    if (isShown) {
      await pause(
          timings.afterCharacterSpeechBubbleBeforeAlternatives
      )
      this.alternativesShown = true
    }
  },
  async playerSpeechBubbleShown (isShown) {
    if (isShown) {
      await pause(timings.afterMessageAlternativeClick)
      this.playerSpeechBubbleShown = false
    } else {
      this.characterSpeechBubbleShown = true
    }
  },
  async card () {
    this.cardTextIndex = 0
  },
  async senderId (newSenderId, previousSenderId) {
    if (
        this.visibleCharacters.includes(newSenderId) &&
        this.visibleCharacters.includes(previousSenderId)
    ) {
      await pause(
          timings.afterCharacterSpeechBubbleBeforeAlternatives
      )
      this.alternativesShown = true
    }
  }
},
mounted () {
  this.onComponentShown()
},
beforeMount () {
  this.$store.commit(
      'componentVisibility/shown',
      'channel-message'
  )
},
```

```
  beforeDestroy () {
    this.$store.commit(
        'componentVisibility/hidden',
        'channel-message'
    )
  },
  methods: {
    async onComponentShown () {
      this.playerSpeechBubbleShown = false
      this.alternativesShown = false
      await pause(timings.beforeInitialCharacterSpeechBubble)
      this.characterSpeechBubbleShown = true
    },
    async onAlternativeClick (alternative) {
      if (isAlternativeContentTrivial(alternative)) {
        const senderStaysTheSame = (
            this.card.sender ===
            this.cardOfAlternative(alternative).sender
        )
        const alternativeCardIsExit =
            this.cardOfAlternative(alternative).effects
                .find(({ functionName }) =>
                    functionName === 'exit'
                )
        this.alternativesShown = false
        this.$emit('alternative-click', alternative)
        if (senderStaysTheSame && !alternativeCardIsExit) {
          await pause(
              timings.afterCharacterSpeechBubbleBeforeAlternatives
          )
          this.alternativesShown = true
        }
      } else {
        this.playerSpeechBubbleShown = true
        this.characterSpeechBubbleShown = false
        this.alternativesShown = false
        this.$emit('alternative-click', alternative)
      }
    },
    async progressDoubleBubble () {
      this.$playSound.showDoubleBubbleClick()
      this.cardTextIndex++
      this.alternativesShown = false
      await pause(
          timings.afterCharacterSpeechBubbleBeforeDoubleBubbleButton
      )
      this.alternativesShown = true
    },
    progressTutorial () {
      this.$playSound.showDoubleBubbleClick()
      this.$store.dispatch('tooltip/progressTutorial')
    }
  }
}
</script>
```

Bilaga 2: Exempel på kombinationen av reaktiviteten i Vue och både anpassade egenskaper (eng. *custom properties*) och `calc()` i CSS

```vue
<template>
  <div
      ref="speechBubble"
      class="speech-bubble"
      :class="{} /* ... */"
  >
    <!-- <div class="shadow"> ... </div> -->
    <!-- <div class="emboss"> ... </div> -->
    <!-- <div class="foreground"> ... </div> -->
    <div class="content">
      <p
          :style="{
          hyphens: (
            text && (text.length > 30)
              ? 'auto'
              : 'manual'
          )
        }"
          v-html="speechBubbleText"
      />
    </div>
  </div>
</template>

<script>
export default {
  props: {
    text: {
      type: String,
      default: null
    }
    // ...
  },
  // data () { ... }
  computed: {
    speechBubbleText () {
      const speechBubbleText = this.text?.trim()
      if (speechBubbleText && !speechBubbleText.match(/[.!?]$/)) {
        return speechBubbleText + '.'
      } else {
        return speechBubbleText
      }
    }
  },
  watch: {
    text () {
      this.updateTextLengthProperty()
      // ...
    }
  },
```

```
  mounted () {
    this.updateTextLengthProperty()
  },
  methods: {
    updateTextLengthProperty () {
      if (this.text?.length > 0) {
        this.$refs?.speechBubble.style.setProperty(
            '--text-length',
            this.text.length
        )
      }
    }
  }
}
</script>

<style scoped>
/* ... */
.speech-bubble {
  /* `--base` och `--ru-base` är definierade
   * högre upp i hierarkin. */
  --text-length: 0;
  --font-size: max(
      calc(var(--base) * 0.6),
      calc(
          0.8
          * var(--base)
          * 100
          / (100 + var(--text-length))
      )
  );
  --text-field-width: min(
      calc(7 * var(--base)),
      calc(
          4
          * var(--base)
          * (150 + var(--text-length))
          / 150
      )
  );
  --text-field-min-height: min(
      calc(6 * var(--base)),
      calc(
          2.8
          * var(--base)
          * (100 + var(--text-length))
          / 100
      )
  );
}
.speech-bubble .content p {
  font-size: var(--font-size);
  width: var(--text-field-width);
  min-height: var(--text-field-min-height);
  /* ... */
}
</style>
```

Bilaga 3: Utdrag ur Vuex-moduler i *Taloussankari Junior*

`index.js` :

```
// import ...
const localStorageName = 'tsj-data'

export const state = () => ({
  game: null,
  // ...
})

export const getters = {
  chapterIdList: state =>
    state.game.chapter.map(({ id }) => id),
  chapterIndex: (state, getters) =>
    chapterId =>
      getters.chapterIdList
        .findIndex(id =>
          (id === chapterId)
        ),
  // ...
  isAllStateEmpty (state, getters) {
    const gettersToCheck = [
      // '.../isStateEmpty'
    ]
    return gettersToCheck
      .every(getter => getters[getter])
  },
  dataToSave: state => ({
    // ...
    money: state.money,
    // ...
  }),
  hasSavedGameSessionInLocalStorage () {
    return window.localStorage.getItem(localStorageName)
  },
  // …

  /**
   * Money getters
   */
  money:
    state =>
      chapterId =>
        state.money[chapterId].current,
  maxMoney:
    state =>
      chapterId =>
        state.money[chapterId].maximum,
  cumulativeMoney:
    (state, getters) =>
      chapterId =>
        getters.chapterIdList
          .slice(0, (getters.chapterIndex(chapterId) + 1))
          .map(id => getters.money(id))
          .reduce((acc, cur) => acc + cur, 0),
```

```
  maxCumulativeMoney:
    (state, getters) =>
      chapterId =>
        getters.chapterIdList
          .slice(0, (getters.chapterIndex(chapterId) + 1))
          .map(id => getters.maxMoney(id))
          .reduce((acc, cur) => acc + cur, 0),
  totalMoney:
    (state, getters) =>
      getters.chapterIdList
        .map(chapterId =>
          getters.money(chapterId)
        )
        .reduce(
          (acc, cur) =>
            acc + cur,
          0
        ),
  maxTotalMoney:
    (state, getters) =>
      getters.chapterIdList
        .map(chapterId =>
          getters.maxMoney(chapterId)
        )
        .reduce(
          (acc, cur) =>
            acc + cur,
          0
        ),
  // ...
}

export const mutations = {
  setGameData (state, { game }) {
    state.game = game
  },
  // ...
}

export const actions = {
  // ...
  resetAllState ({ state, dispatch, commit }) {
    const game = state.game
    // ...
    dispatch('money/initializeAllChapters', { game })
    // ...
  },
  async saveGameSessionToFirestore ({ state, commit, getters }) {
    if (getters.isAllStateEmpty) {
      return {
        error: 'no data'
      }
    }
    // ...
    await this.$fire.firestore
      // ...
      .set({
        ...getters.dataToSave,
        // ...
      })
    // ...
  },
```

```
  restoreSession ({ dispatch }, { data }) {
    const {
      // ...
      money,
      // ...
    } = data
    // ...
    dispatch('money/restoreState', { stateToRestore: money })
    // ...
  },
  async loadSavedGameSessionFromFirestore (
    { dispatch, commit },
    { firestoreCode }
  ) {
    const doc = await this.$fire.firestore/* ... */.get()
    if (doc.exists) {
      // ...
      dispatch('restoreSession', { data: doc.data() })
      // ...
    }
  },
  saveGameSessionToLocalStorage ({ getters }) {
    window.localStorage.setItem(
      localStorageName,
      JSON.stringify(getters.dataToSave)
    )
  },
  loadSavedGameSessionFromLocalStorage ({ dispatch }) {
    const data = JSON.parse(
      window.localStorage.getItem(localStorageName)
    )
    if (data) {
      dispatch('restoreSession', { data })
    }
  },
  // ...
}
```

`money.js` :

```
export const state = () => ({})

export const getters = {
  total:
    state =>
      Object.entries(state).reduce(
        (acc, [id, { current }]) =>
          acc + current,
        0
      ),
  maximumTotal:
    state =>
      Object.entries(state).reduce(
        (acc, [id, { maximum }]) =>
          acc + maximum,
        0
      )
}
```

```
export const mutations = {
  initializeForChapter (state, { chapterId, current = 0, maximum }) {
    this._vm.$set(state, chapterId, {})
    this._vm.$set(state[chapterId], 'current', Number(current))
    this._vm.$set(state[chapterId], 'maximum', Number(maximum))
  },
  set (state, { chapterId, value }) {
    state[chapterId].current = Number(value)
  },
  add (state, { chapterId, value }) {
    state[chapterId].current += Number(value)
  }
}

export const actions = {
  initializeAllChapters ({ commit }, { game }) {
    game.chapter.forEach(({ id, maxMoney }) => {
      commit('initializeForChapter', { chapterId: id, maximum: maxMoney })
    })
  },
  restoreState ({ commit }, { stateToRestore }) {
    Object.entries(stateToRestore)
      .forEach(([chapterId, { current }]) =>
        commit('set', { chapterId, value: current })
      )
  }
}
```