Verslag Tinlab Advanced Algorithms

Tommie Terhoeve
0926280

April 12, 2019

Contents

1 Inleiding

Dit verslag dient als leerverslag om zo mijn progressie gedurende het vak bij te houden. Tevens is dit verslag bedoeld als samenvatting van de leerstof zodat in de toekomst dit verslag gelezen kan worden als opfrissing van de stof.

In dit verslag zullen de diverse onderwerpen van het vak belicht worden.
Voor het ontwerpen van systemen zijn er systematische methodes ontwikkelt om dit zo efficiënt mogelijk aan te pakken. Een van de meest gebruikte methodes is Agile [1]. Een andere populaire methode is de waterval methode [2].

2 Requirements

2.1 Requirements

De term requirements kan op een aantal manieren gedefinieerd worden.
Requirements kunnen vanuit het organisatorische en software-perspectief bekeken worden. Het organisatorische persectief wordt ook wel de system requirements genoemd. Het software-perpesctief staat ook bekend als de software-specifications.

Bij het opstellen van system requirements wordt er gekeken naar de specifieke requirements van de stakeholders voor het desbetreffende systeem. Hierbij worden puur de requirements opgesteld van het gewenste eindresultaat. Bij het opstellen van System-requirements wordt er over het algemeen nog geen gebruik gemaakt van technische jargon. System requirements kunnen gezien worden als een manier om de verandering van het oude naar het nieuwe systeem te beheren. Bijvoorbeeld bij de vernieuwing van een administratiesysteem zullen de stakeholders bepaalde functionaliteiten willen die de oude versie niet bevatte. [3]

In de mainstream computer wereld worden system requirements ook omschreven als hardware requirements om bepaalde software te draaien. Op deze manier kunnen de ontwikkelaars garanderen dat de desbetreffende software werkt op bepaalde hardware [4].

2.2 specificaties

Specificaties kunnen opgedeeld worden in functionele en niet-functionele requirements. Functionele requirements worden gespecificeerd door:

1. input
2. output
3. processing

Functionele requirements omschrijven de fundamentele functies van de software voor het systeem. Bijvoorbeeld wat het systeem moet doen bij een onverwachte situatie. Niet-functionele requirements zijn extra requirements die beperkingen leggen op het systeem. Denk hierbij aan eisen die te maken hebben met veiligheid, gebruiksvriendelijkheid of uitbreidbaarheid. Bijvoorbeeld: Doordat het systeem binnen 10 seconden moet opstarten zullen bepaalde processen niet opgestart worden tijdens het opstarten van het systeem.

Vaak gebeurt het dat niet-functionele requirements uiteindelijk toch geclassificeerd worden als functionele requirements. Hierdoor hebben sommige de voorkeur om deze termen niet te gebruiken in de praktijk[3].

2.3 Mode confusion

Mode confusion is een fenomeen waarbij het systeem zich anders gedraagt dan dat de gebruik verwacht. Deze mode confusion scenario's kunnen onschuldig zijn, bijvoorbeeld wanneer je telefoon zich opnieuw willekeurig opstart. Uiteraard in scenario's zoals ziekenhuizen of vliegtuigen kan dit zorgen voor levensgevaarlijke scenario's. [5]

2.4 Scenario's

Vlucht 1951 Turkish Airlines

Op 25 februari 2009 crashte vlucht Turkish Airlines 1951 1,5km voor de landingsbaan. Hierbij kwamen 9 personen om het leven. Vliegtuigongelukken hebben vrijwel altijd meerdere oorzaken. Tijdens het onderzoek zijn ze tot de conclusie gekomen dat de meest voorhandliggende oorzaak een combinatie is van een menselijke en instrumentale fout.

Doordat de hoogtemeter niet correct functioneerde gedraagde de automatische piloot zich anders dan verwacht. Dit is een typisch geval van mode confusion. Het systeem gedraagde zich anders dan dat de piloten verwachtte waardoor er verwarring ontstond in de cockpit.

Therac-25

De Therac-25 was een radiatietherapie systeem dat zorgde voor 4 doden door een overdosis radiatie tussen juni 1985 en januari 1987. Bij het onderzoek kwam tot het licht dat er een groot aantal software-fouten in het systeem zaten. Deze software werd door 1 persoon geschreven. Vooral op het gebied van veiligheid zaten er veel gebreken in de software. Dit zou voorkomen kunnen worden indien er concrete niet-functionele specificaties werden opgesteld.

Tevens speelt hier ook mode-confusion mee aangezien er verwarring was bij de gebruikers tijdens het geven van de radiatietherapie aangezien het systeem onverwachts reageerde [6].

Tsjernobyl 1986

Op 26 april 1986 ontplofte de nucleaire reactor van de Tsjernobyl kerncentrale. De oorzaak hiervan was een combinatie van slecht design, regulatie en management. Zo kon de reactor buiten de veilige specificaties gedraaid worden. Dit is een voorbeeld van slechte niet-functionele eisen aangezien er niet genoeg is nagedacht over de veiligheid van de reactor [7].

ARIANE 5 vlucht 501

Op 4 juni 1996 was de eerste vlucht van de Ariane 5 raket. Na 40 seconde week de raket af van de geplande route en explodeerde doordat de self-destruction werd geactiveerd. De oorzaak hiervan was een verlies aan guidance en hoogte informatie. Deze informatie was verloren door diverse fouten in de software door een gebrek aan concrete software specifications [8].

3 Modellen

3.1 De Kripke structuur

De Kripke structuur is een variatie van de state transition diagram. Dit word gebruikt voor model checking om zo het gedrag van een systeem te simuleren. Dit kan dus gebruikt worden om systemen te testen op eventuele logische fouten [9].

Kripke structuren bestaan voornamelijk uit states en transities. Een state kan omschreven worden als een staat waar het systeem zich in kan bevinden. Bijvoorbeeld een lamp heeft 2 states: aan en uit. De overgang tussen states wordt de transitie genoemd. De state waarin het systeem bevindt tijdens het opstarten wordt de initial state genoemd. Een Kripke structuur wordt gedefinieerd als een tuple M = (S,I,T,L)

- S: een verzamling states (s0,s1,s2,sx)
- I: een verzameling van de initial states (i0,i1,ix)
- T: een lijst van transities tussen de states (t1,t2,tx)
- L: de labelingsfunctie die elke state een label geeft met proposities die waar zijn voor die state [10].

3.2 Soorten modellen

Behalve Kripke structuren zijn er ook andere soorten modellen. Zo is er de labeled transition system (LTS). Kripke structuren en LTS lijken zeer erg op elkaar en zijn praktisch verwisselbaar [11].
Een ander soort model is het Timed Transition System (TTS). het TTS model lijkt ook zeer op Kripke structuren. Het grootste verschil hierbij is dat er in de tuple van de TTS ook clock variabele zijn inbegrepen die de tijd bijhouden [12].

3.3 Tijd

In het eerder benoemde TTS model wordt er niet gebruik gemaakt van reguliere tijdseenheden zoals seconden, minuten uren etc. In plaats daarvan wordt er gebruik gemaakt van "tijdseenheden". Tijd wordt in UPPaal net als in het TTS model bijgehouden door middel van clock variabele [13].

3.4 Guards en invarianten

Guards zijn condities die gelden in een bepaalde transitie. Bijvoorbeeld : Indien x $==$ 2 kan de transitie van state A naar B plaats vinden. Invarianten zijn condities die gelden in een bepaalde state. Een voorbeeld van een invariant is: y $>$ 3 anders kan het systeem zich niet in deze state bevinden [13].

3.5 Deadlock

Een deadlock in een computersysteem vind plaats wanneer er één of meerdere processen geblokkeerd worden doordat de eisen voor die processen nooit of niet meer behaald kunnen worden. Hierdoor kan het systeem zich voor altijd in één enkel proces blijven bevinden [14]. In een Kripke structuur kan dit gezien worden als een state die niet kan transitioneren naar een andere state, aangezien alle invarianten van de aangesloten states niet behaald kunnen worden.

3.6 Zeno gedrag

Zeno gedrag komt voor in systemen wanneer er een oneindig aantal transities in een eindige tijd plaatsvind. Een bekend voorbeeld hiervan in de filosofie is Achilles en de schildpad. De theorie is dat Achilles nooit de schildpad in kan halen indien de schildpad een voorsprong heeft. Achilles zal dus in een eindige tijd een oneindig aantal stappen zetten zonder dat hij de schildpad inhaalt [15].

Bij het ontwerpen van systemen is het belangrijk om rekening te houden met Zeno gedrag. Bijvoorbeeld indien een sensor een bepaalde meting moet uitvoeren kan je Zeno gedrag voorkomen door een timer te gebruiken die de sensor om de x aantal seconden laat meten [16].

4 Logica

4.1 Propositielogica

Propositielogica is een tak van logica die werkt met proposities. Proposities zijn uitspraken die waar of onwaar zijn. Bijvoorbeeld: "Het regent" kan waar of onwaar zijn.
Doormiddel van logical operators kunnen er gecombineerde proposities gemaakt worden. Bijvoorbeeld: "Het sneeuwt en het is koud". De logical operator in dit geval is het woord "en", dit is een conjunctie en wordt aangeven met het symbool $\wedge$. Dit zijn de meest voorkomende logische operators [17]:

- $\wedge$ (AND): Conjunctie
- $\vee$ (OR): Disjunctie
- $\oplus$ (XOR) : Inclusieve disjunctie
- $\neg$ (NOT): Negatie, $\neg$p betekent dat p niet waar is.
- $\rightarrow$: Implicatie, p $\rightarrow$ q betekent: Als p waar is dan is q ook waar.

4.2 Predicatenlogica

Predicatenlogica is een uitbreiding van de propositie logica. Net zoals in de propositielogica wordt er gebruik gemaakt van proposities en logische operators. In de predicatenlogica zijn er ook predicaten en kwantoren. Predicaten zeggen iets over een object of persoon. Bijvoorbeeld: Tom is een kat. Kwantoren worden gebruikt om te definiëren of een statement altijd waar is, soms waar is of nooit waar is.
Aan een predicaat wordt altijd een naam gegeven met argumenten. Vaak wordt hiervoor enkel 1 letter gebruikt voor de naam en argumenten. Bijvoorbeeld: Het regent vandaag en de zon schijnt. Dit predicaat zou je op kunnen schrijven als W(r,z) [18].

4.3 Kwantoren

In de predicatenlogica zijn er twee kwantoren:

- $\forall$ Universele kwantor: Dit betekent voor alle.
- $\exists$ Existentiële kwantor: Dit betekent er is of er bestaat.

Bijvoorbeeld: Iedereen heeft een fiets. Dit kan je schrijven als $\forall$x F(x). x is hierbij de variabele die mensen voorstelt. Als we willen zeggen dat sommige mensen een fiets hebben, dan zouden we het schrijven als: $\exists$x F(x) [18].

4.4 Dualiteiten

Dualiteiten zijn in de logica formules of relaties die uiteindelijk hetzelfde betekenen. Twee bekende voorbeelden in de propositielogica zijn de wetten van De Morgan. Voor twee proposities P en Q gelden de volgende wetten [19]:

- $\neg$ (P$\wedge$Q) = ($\neg$P $\vee$ $\neg$Q)
- $\neg$ (P$\vee Q$) = ($\neg$P $\wedge$ $\neg$Q)

Op deze manier zijn er ook diverse voorbeelden van dualiteiten uit de temporele logica. Temporele logica wordt gebruikt voor het controleren van Kripke structuren in applicaties zoals UPpaal. De volgende operatoren zullen later in dit verslag uitgelegd worden. [20]

- AX(f) = $\neg$EX($\neg$f)
- AG(f) = $\neg$EF($\neg$f)

5 Computation tree logic

Computation tree logic, ook wel CTL* genoemd is een vorm van logica waarmee systemen kunnen worden gecontroleerd op logische fouten. CTL* formules beschrijven eigenschappen van de computation tree. CTL* formules bestaan uit path quantifiers (A,E) en temporal operators (G,F,X). De path quantifiers worden gebruikt om te controleren of een bepaald pad of alle paden vanuit een bepaalde state de bepaalde eigenschap bevatten. De temporal operators kunnen specifieker controleren waar en hoe vaak de eigenschappen in de paden bestaan. Over het algemeen zijn de eigenschappen proposities die getest kunnen worden. Zo kan er met CTL* getest worden of in het systeem een bepaalde propositie waar is gedurende een heel pad in de computation tree [20].

5.1 De computation tree

De computation tree is een visualisatie van het logische model. In de computation tree worden de logische stappen tussen states en transitions beschreven. In het geval van totale modellen zal de computation tree oneindig doorlopen aangezien er oneindig veel logische stappen mogelijk zijn. In een Kripke structuur is de "root" state waar de computation tree begint de initial state van de Kripke structuur tuple [20].

5.2 Operator: AG

De operator AG in CTL* staat voor All Globally.
Het model voldoet aan AG(P) wanneer in elk pad, elke state propositie P waar is [20].

5.3 Operator: EG

De operator EG staat voor Eventually Globally.
Het model voldoet aan EG(P) wanneer in 1 of meer paden elke state propositie P waar is [20].

5.4 Operator: AF

De operator AF staat voor All Finally.
Het model voldoet aan AF(P) wanneer in elk pad, bij tenminste 1 state Propositie P waar is [20].

5.5 Operator: EF

De operator EF staat voor Eventually Finally.
Het model voldoet aan EF(P) wanneer in tenminste 1 pad, bij tenminste 1 state propositie P waar is [20].

5.6 Operator: AX

De operator AX staat voor All Next.
Het model voldoet aan AX(P) wanneer in de eerste state vanaf elk pad die aftakt van de initial state propositie P waar is [20].

5.7 Operator: EX

De operator EX staat voor Eventually Next.
Het model voldoet aan EX(P) wanneer in tenminste 1 pad die aftakt van de initial state, in de eerste state propositie P waar is [20].

5.8 Operator: p U q

De operator p U q staat voor propositie p Untill propositie q.
een pad in een model voldoet aan p U q wanneer propositie p waar totdat q waar is [20].

5.9 Operator: p R q

De operator p R q staat voor propositie p Release propositie q.
Een pad voldoet aan p R q wanneer propositie p waar is totdat q waar is. Het verschil met p U q is dat bij p R q propositie p en q beide waar zijn in de laatste state waar p waar is [20].

5.10 Fairness

Fairness is de geneste operator AG(AF(P)).
Het model voldoet aan deze geneste operator indien je vanuit elk pad op een gegeven moment p = waar tegenkomt [20].

5.11 Liveness

Liveness is de geneste operator AG(p $\rightarrow$ AF(q)).
Het model voldoet aan deze geneste operator indien propositie q in elk pad op een gegeven moment geldt als p geldt [20].

References

[1] A. Manifesto, "Manifesto for agile software development," 2001.

[2] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, no. 5, pp. 61–72, 1988.

[3] P. Loucopoulos and V. Karakostas, *System Requirements Engineering*. 01 1995.

[4] "System requirements." https://en.wikipedia.org/wiki/System$_requirements, n.d.$

[5] A. L. Jan Bredereke, *A Rigorous View of Mode Confusion*. 09 2002.

[6] N. G. Leveson and C. S. Turner, "An investigation of the therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[7] A. Shlyakhter and R. Wilson, "Chernobyl: the inevitable results of secrecy," *Public Understanding of Science*, vol. 1, no. 3, pp. 251–259, 1992.

[8] J.-L. Lions, L. Luebeck, J.-L. Fauquembergue, G. Kahn, W. Kubbat, S. Levedag, L. Mazzini, D. Merle, and C. O'Halloran, "Ariane 5 flight 501 failure report by the inquiry board," 1996.

[9] "Kripke structure (model checking)." https://en.wikipedia.org/wiki/Kripke$_structure_(model_checking), n.d.$

[10] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *International conference on tools and algorithms for the construction and analysis of systems*, pp. 193–207, Springer, 1999.

[11] R. Schoren, "Correspondence between kripke structures and labeled transition systems for model minimization,"

[12] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi, "Minimization of timed transition systems," in *International Conference on Concurrency Theory*, pp. 340–354, Springer, 1992.

[13] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal 4.0,"

[14] R. C. Holt, "Some deadlock properties of computer systems," in *Proceedings of the third ACM symposium on Operating systems principles*, pp. 64–71, ACM, 1971.

[15] "Zeno's paradoxes." https://en.wikipedia.org/wiki/Zenon.d.

[16] A. D. Ames, A. Abate, and S. Sastry, "Sufficient conditions for the existence of zeno behavior," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 696–701, IEEE, 2005.

[17] M. L. Seymour Lipschutz, *Schaum's Outlines Discrete Mathematics*. 2009.

[18] J. K. Mattila, "Predicate logic,"

[19] W. Oele, "Logica." https://med.hr.nl/oelew/logica/logica.pdf).

[20] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.