CHALMERS | GÖTEBORGS UNIVERSITET

Förbättring av ett webbaserat språkverktyg

Kandidatarbete inom data- och informationsteknik

GUSTAV EHRENBORG

FREDRIK ESTELIUS

FREDRIK HIDSTRAND

ANDREAS MATHISEN

ROBIN PETTERSSON

MAGNUS SJÖQVIST

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Förbättring av ett webbaserat språkverktyg

GUSTAV EHRENBORG

FREDRIK ESTELIUS

FREDRIK HIDSTRAND

ANDREAS MATHISEN

ROBIN PETTERSSON

MAGNUS SJÖQVIST

Examiner: SVEN-ARNE ANDRÉASON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Sammanfattning

Avdelningen för fackspråk och kommunikation vid Chalmers använder sig av ett webbaserat språkverktyg, *EngOnline*, för att undervisa engelsk grammatik. Språkverktyget lider av en del brister. Dessa innefattar bland annat stabilitetsproblem och ett föråldrat gränssnitt, men framförallt saknas det dokumentation. Eftersom dokumentation saknas och då EngOnline är i behov av underhåll så har detta lett till att *Avdelningen för fackspråk och kommunikation* vill ersätta språkverktyget med ett nytt och väldokumenterat sådant.

Utvecklingen av det nya språkverktyget har skett med en agil arbetsprocess; detaljerad dokumentation ersattes med en effektiviserad kommunikation. Dessutom har kontinuerliga möten med kunden, domänens experter, gett feedback genom utvecklingsprocessen.

Projekten resulterade i ett nytt och dokumenterat verktyg som har ett nytt och modernt gränssnitt. Implementeringen skedde i Django, ett ramverk för webbutveckling baserat på Python. Verktyget erbjuder den funktionalitet som prioriterades högst tillsammans med kunden. Från det gamla EngOnline migrerades merparten av innehållet till det nya verktyget.

Abstract

The Division of Language and Communication at Chalmers uses a web-based language tool, named EngOnline, to teach English grammar. The language tool suffers from some shortcomings. These include stability problems and an outdated interface but, above all, there is no documentation. Because of the lack of documentation and EngOnline's need of maintenance, the Division of Language and Communication has decided to replace the language tool with a new and well documented one.

The new language tool was developed using an agile work process; detailed documentation was replaced with more efficient communication. In addition, regular meetings with the client, the experts of the domain, provided feedback throughout the development process.

The project resulted in a new and more documented tool with an improved and modernized user interface. The implementation was done in Django, a Python web framework. The new language tool provides the functionality that was prioritized during meetings with the client. The majority of the content from the original EngOnline was migrated to the new language tool.

Ordlista

AJAX	*Asynchronous JavaScript and XML,* Teknik för att kommunicera asynkront med en webbapplikation
App	En modul i ett Djangoprojekt
CID	*Chalmers-id,* Personligt användarkonto för studenter på Chalmers
Django	Ramverk för webbutveckling baserat på Python
EngOnline	Namnet på språkverktyget
ER-diagram	Diagram för att beskriva databasmodeller
Fackspråk	Avdelningen för fackspråk och kommunikation, kunden
HTML	*Hyper Text Markup Language,* ett märkspråk för webbapplikationer
JavaScript	Skriptspråk som kan processeras i webbläsaren på klientsidan
ITS	*Chalmers IT-support,* ansvarar för drift och underhåll av centrala IT-system på Chalmers
MVC	*Model View Controller,* ett designmönster för mjukvarusystem
ORM	*Object Relational Mapping*, teknik för arbete med en objektrepresentation för att ändra och läsa från en relationsdatabas
Python	Programmeringsspråk som systemet huvudsakligen utvecklats i
UML	Unified Modeling Language, ett modelleringsspråk

Förord

Den här rapporten behandlar kandidatarbetet *Förbättring av ett webbaserat språkverktyg* som genomfördes under våren 2013 vid Institutionen för data- och informationsteknik på Chalmers. Kandidatarbetet utfördes på uppdrag av Avdelningen för fackspråk och kommunikation. Projektgruppen önskar tacka Andreas Eriksson och Linda Bradley på Fackspråk för deras bidrag till projektet och hjälp med rapporten. Vi vill också tacka vår handledare Sven-Arne Andréasson för den hjälp vi har fått.

Innehållsförteckning

1. Inledning

Avdelningen för fackspråk och kommunikation, hädanefter *Fackspråk*, vid *Chalmers* har sedan början av 2000-talet använt sig av språkverktyget *EngOnline* för att utbilda studenter i engelsk grammatik. Språkverktyget är en webbaserad lösning som har gjort det möjligt för studenter vid *Chalmers* att använda sin webbläsare för att ta del av kursinnehåll om engelsk grammatik. EngOnline tillåter studenterna att ta del av material som lär ut teorikunskaper i ämnet. Samtidigt som de läser in teori finns det också möjligheter att göra övningar och att testa sina kunskaper dels med kapiteltest, dels med en diagnos som testar studenten på hela det tillgängliga materialet.

Språkverktyget har fått visst underhåll och en uppdatering av gränssnittet gjordes under år 2004. Trots detta finns det en del stabilitetsproblem, det kan exempelvis hända att användare loggas ut om för många andra loggar in. I vissa fall fungerar inte heller den centrala inloggningen som används vid Chalmers, och en administratör för verktyget måste då manuellt lägga till lokala användare.

EngOnline är i behov av ytterligare underhåll för att åtgärda stabilitetsproblemen. Det finns dessutom en önskan från Fackspråk om utökad funktionalitet och ett modernare gränssnitt. Underhåll och utbyggnad är dock något som är mycket svårt att genomföra eftersom dokumentation för verktyget saknas. Problemet förvärras dessutom av att den som tidigare varit ansvarig för verktyget inte längre jobbar kvar.

Detta har lett till att Fackspråk vill ersätta EngOnline med ett nytt och väldokumenterat språkverktyg som är både lätt att underhålla och att bygga ut. Det nya språkverktyget behöver ha ett modernare utseende. Gränssnittet bör vara funktionellt och tilltalande för att locka studenter att använda det för att förbättra sin engelska grammatik.

1.1 Syfte

Projektet syftar till att kartlägga och utveckla en produkt som ska kunna användas av Fackspråk vid Chalmers för att utbilda studenter i engelsk grammatik. Produkten är ett webbaserat verktyg för språkinlärning som ska ersätta det nuvarande systemet EngOnline. Verktyget ska dock återanvända det nuvarande systemets innehåll gällande sidor och övningar.

1.2 Avgränsningar

En del av problematiken kring att skapa ett verktyg som ska undervisa är att gränssnittet ska bli pedagogiskt. Ett pedagogiskt gränssnitt är dock inget som kommer att ges någon särskild hänsyn under utvecklingen. Ett huvudmål med systemet är dock att det ska vara utbyggbart och lätt att underhålla vilket innebär att det blir möjligt att i framtiden utveckla de pedagogiska aspekterna av systemet och ge dessa fokus i efterhand.

2. Metod

Detta kapitel handlar om hur det förberedande arbete gick till och den arbetsprocess som användes för att framställa verktyget. Här beskrivs planering, informationsinsamling och implementering.

2.1 En agil arbetsprocess

Projektet har genomförts *agilt*. Detta innebär bland annat att fokus har flyttats från utförliga designdokument, så som detaljerade sekvensdiagram, och istället har arbetet utförts i samlad grupp. Arbetet delades också upp i iterationer där fler krav successivt infördes per iteration enligt prioritering. Flertalet möten med kund hölls sedan under utvecklingen för att få respons genom hela utvecklingsprocessen. Dessa regelbundna möten innebar också att kravställning och domänanalys kunde växa fram och förfinas gradvis. Detta är viktigt för att undvika missförstånd och därmed kunna leverera den produkt som kunden vill ha.

Under arbetets gång har varje vecka medfört minst två projektmöten, under dessa har scrum-liknande möten använts. Detta har inneburit att varje projektmedlem i början av mötet har fått berätta vad denne jobbar med för tillfället, vilka problem som är aktuella i samband med detta och hur man ska gå vidare med arbetet.

Den agila metoden sänker kostnaden för informationsutbyte mellan projektmedlemmar och minskar tiden mellan beslut och resultat (Cockburn och Highsmith, 2001). Detta uppnås enligt Cockburn och Highsmith (2001) genom att placera projektmedlemmar på samma plats och ersätta designdokument med direktkommunikation. Dessutom understryker de att användare, som är experter på domänen, ska finnas tillgängliga under utvecklingen så att projektgruppen kan få relevant feedback och utplåna missförstånd så tidigt som möjligt. Det innebär alltså att regelbundna möten med kund kan leda till att missuppfattningar gällande funktionalitet och design kan upptäckas tidigt och därmed också åtgärdas tidigt. Detta ökar därmed sannolikheten att kunden får den produkt som de vill ha.

2.2 Planering och tidsplan

I samband med projektets uppstart gjordes en planeringsrapport. Centralt under denna fas var att bygga upp en initial bild av problemområdet. En tidsplan togs fram där arbetets iterationer infördes.

Tidsplanen som Gantt-schema återfinns i *Appendix A - Tidsplan*.

2.3 Kravställning och analys av problemområde

Initialt behövde projektet kravställas och problemområdet undersökas. Detta var något som skedde parallellt och i samarbete med kund. Möte med kunden anordnades så tidigt som möjligt för att få en förståelse för dennes behov. Endast genom en förståelse för användarna är det möjligt att ta fram en bra kravställning (Rogers, Sharp och Preece 2011, s. 355).

I samband med framtagandet av kravställning skrevs också kortfattade användningsfall. Eftersom flertalet krav kan fångas av ett användningsfall kan dessa användas för att gruppera

kraven. Detta utnyttjades genom att tillsammans med kunden prioritera kraven med hjälp av användningsfallen. På detta vis kunde flera krav prioriteras samtidigt och mängden jämförelser och beslut minskade därför till en mer hanterbar nivå. Dessutom, enligt Larman (2004, s. 64), är användningsfall ett bra sätt att kommunicera med kund eftersom de är användarorienterade och komplexiteten enkelt kan skalas efter behov. Larman (2004, s. 92) menar vidare att användningsfall förenklar kommunikation kring krav genom att sätta dessa i ett sammanhang av ett typiskt användningsscenario.

Prioriteten som tilldelades kraven delades upp i prioritet ett, två och tre. Prioritet ett var funktionalitet som var absolut nödvändig, prioritet två var funktionalitet som behövdes och slutligen var prioritet tre funktionalitet som skulle implementeras i mån av tid. Den sista kategorin tillåts innehålla funktionalitet som inte behöver implementeras inom projektets tidsram. Detta gör att funktionalitet som är viktig i framtiden kan placeras här och det blir då naturligt att designa systemet så att det sedan tillåter dessa utbyggnader.

Funktionaliteten implementerades sedan utifrån kravens prioriteringar. Fokus låg i början på att skapa ett gränssnitt med den mest grundläggande funktionaliteten som sedan kunde utökas. Detta gjorde det möjligt att få kundens feedback på det faktiska systemet. Denna återkoppling i samband med användningsfallen gav stöd för en användarorienterad design.

Som stöd för analysen av problemområdet gjordes en domänmodell i modelleringsspråket *UML*. Denna innehåller koncept som är relevanta i domänen och beskriver deras relationer till varandra. Enligt Larman (2004, ss. 134-136) är domänmodellen det mest väsentliga verktyget i en objektorienterad analys. Detta tillskriver han bland annat att en objektorienterad domänmodell minskar steget mellan den mentala modellen och mjukvarumodellen.

Den abstrakta domänmodellen översattes till ett *ER-diagram* som beskriver databasmodellen, en realisering av domänmodellen.

2.4 Undersökning av möjliga hjälpmedel

Tidigt i projektet genomfördes en undersökning av möjliga hjälpmedel. I denna undersöktes ett antal programmeringsspråk, ramverk för webbutveckling och databashanterare. Krav som ställdes på det nya verktyget, att det ska vara framtidssäkert och lätt att underhålla, hamnade i fokus. Undersökningen genomfördes genom att titta på vilka programmeringsspråk som är populära idag och vilka lösningar för webbutveckling dessa erbjuder.

Det primära målet för undersökningen var att så snabbt som möjligt göra ett bra val. Ett bra val innebär här att tekniken har stöd för att skilja på utseende och logik. Dessutom ska det finnas stöd för aktuella lagringsrealiseringar och att tekniken kommer att vara aktuell i en överskådlig framtid. Ett snabbt val var nödvändigt för att kunna implementera majoriteten av utsatt funktionalitet inom tidsramen för projektet.

2.5 Implementering och versionshantering

Under implementeringen användes *Google Code* som ärendehanteringssystem. Alla användningsfall lades in som ärenden med sin givna prioritet. Ärenden fördelades sedan mellan utvecklare under projektets gång.

För att flera personer ska kunna arbete med utvecklingen av verktyget samtidigt användes versionshanteringssystemet *Git*. Git har en central förvaringsplats för koden som alla medlemmar i projektet laddade upp sin kod till. Denna förvaringsplats finns integrerat i Google Code vilket gör att öppna ärende enkelt kan avslutas när kod laddas upp.

2.6 Analys av den gamla databasen

En kopia av databasen från det gamla EngOnline blev tillgänglig vid projektets början och denna analyserades grundligt. Den visade sig innehålla mycket mer än bara data tillhörande EngOnline. Totalt fanns 43 tabeller och endast ett fåtal av dessa verkade vara relevanta för projektet. Ett ER-diagram över de vitala delarna av den gamla databasen återfinns i *Appendix B - ER-diagram för den gamla databasen*.

Första steget för att urskilja relevant data var att analysera tabellernas namn. Bland dessa hittades några namn som verkade mer relevanta än andra. Eftersom en dataförlust skulle vara förrädisk för Fackspråk, då mycket tid och arbete ligger bakom uppsättningen av de uppgifter som finns idag, var det dock viktigt att grundligt gå igenom alla tabeller för att hitta relevant data.

En manuell genomgång av datan i tabellerna gjordes för att identifiera vad de faktiskt innehöll. Då många tabeller innehöll flera tusen rader, som skulle bli alldeles för tidsödande att gå igenom, slumpades ett antal rader fram som kontrollerades. I denna genomgång uppdagades det att databasen innehöll mycket annat än data kopplad till EngOnline. Till exempel innehöll databasens tabell för grammatikfrågor även enkätfrågor om allt från arbetsmiljö till studier. Dessa frågor var ej av värde att behålla. Liknande irrelevant data hittades i merparten av tabellerna.

För själva migreringen skapades en temporär databas för att kunna mellanlagra datan. Ett enkelt program skrevs som tog kontakt med denna databas samt med den gamla databasen. Detta program kördes flera gånger med olika konfigurationer för att extrahera relevant data.

3. Domänanalys

Detta kapitel analyserar och beskriver problemområdet, hur problemet som det nya verktyget ska lösa ser ut. Initialt beskrivs problemområdet och aspekter så som gränssnitt, utvecklingsmiljö, systemdesign, databasdesign, databasmigrering, underhåll och prestanda. Därefter beskrivs domänen med hjälp av UML-diagram. Avslutningsvis behandlas de begrepp som är relevanta för domänen.

3.1 Beskrivning av problemområdet

Fackspråk vill ha ett nytt och väldokumenterat språkverktyg som är enkelt att underhålla och bygga ut. De är nöjda med mycket av den funktionalitet som det gamla EngOnline erbjuder och deras primära önskemål är att det nya språkverktyget ska erbjuda liknande funktionalitet. Det nya verktyget ska dock ha ett mer lättanvänt gränssnitt och möjlighet till utökad funktionalitet.

3.1.1 Användargränssnittet

Studenter vid Chalmers har tillgång till verktyget under hela sin studietid men normalt använder endast en del av studenterna verktyget och då under en åttaveckors läsperiod som en del av undervisningen. Personalen vid Fackspråk som administrerar verktyget använder det däremot över en betydligt längre tid och det är därför önskat att fokus ska ligga på att administrationen ska fungera så smidigt som möjligt. Fackspråk vill att verktyget ska ha ett konsekvent gränssnitt för alla användare vilket innebär att användargränssnittet behöver vara plattformsoberoende.

3.1.2 Utvecklingsspråk och utvecklingsmiljö

Fackspråk ställer inga krav på att ett specifikt utvecklingsspråk eller en specifik utvecklingsmiljö ska användas. Däremot ställs krav på att *Java* inte ska användas på klientsidan och att valet ska vara framtidssäkert. Dessutom behöver önskemålet om att systemet ska vara utbyggbart och lätt att underhålla beaktas.

Att systemet ska vara framtidssäkert och lätt att underhålla innebär att det är viktigt att det finns gott om dokumentation för det ramverk och språk som kommer att väljas. Det är också av största vikt att dessa är tillräckligt etablerade för att finnas kvar inom överskådlig framtid så att systemet inte blir begränsat till ett döende programmeringsspråk som inte kommer stödjas om några år.

3.1.3 System- och databasdesign

Ett av Fackspråks primära mål med projektet är att det nya verktyget ska vara lätt att underhålla och bygga ut i framtiden. Det ställer stora krav på designen av både databasen och systemet. Fackspråk ser ett antal tänkbara scenarion om hur verktyget kan komma att användas i framtiden som måste tas i beaktning när systemet designas. Designen måste uppfylla dagens krav samtidigt som den inte får förhindra framtida utbyggnader och förändringar.

Idag använder Fackspråk endast två typer av användare i verktyget, administratörer och studenter. De är nöjda med detta men då det nya verktyget ska vara utbyggbart och eventuellt användas av andra universitet i framtiden behöver hanteringen av användarklasser göras så dynamisk som möjligt.

För underlätta framtida förändringar är det viktigt att systemet blir uppdelat i tydliga och fristående delar. Om Fackspråk till exempel skulle vilja byta databas i framtiden så ska man kunna gör det utan att behöva göra några större förändringar i det övriga systemet.

3.1.4 Migrering av den gamla databasen

Databasen som tillhör det gamla EngOnline innehåller data som ska föras över till det nya verktyget. Den gamla databasen innehåller flera års data och att manuellt föra över denna skulle ta väldigt mycket tid i anspråk. Processen att exportera data från den gamla databasen och sedan importera den i den nya databasen behöver därför automatiseras.

Det gamla EngOnline med dess utformning och funktionalitet resulterar i en datamodell som kommer att skilja sig från den nya. Detta medför att export och import av data mellan databaserna inte kommer att vara trivialt. Datan i den gamla datamodellen som fortfarande är relevant för den nya måste identifieras och passas in. Detta kan resultera i att data som inte var aktuell att lagra i den gamla modellen måste skapas eller ignoreras. Det senare fallet innebär att implementationen av det nya systemet måste ta hänsyn till att all data inte är helt komplett i alla situationer.

3.1.5 Modulärinloggning

Verktyget ska kunna hantera externa användare och ha stöd för att i framtiden kunna hantera användare från andra universitet. Detta ställer krav på autentiseringssystem och även på gränssnittet som måste inkludera val av inloggningssystem, om detta inte kan ske automatiskt.

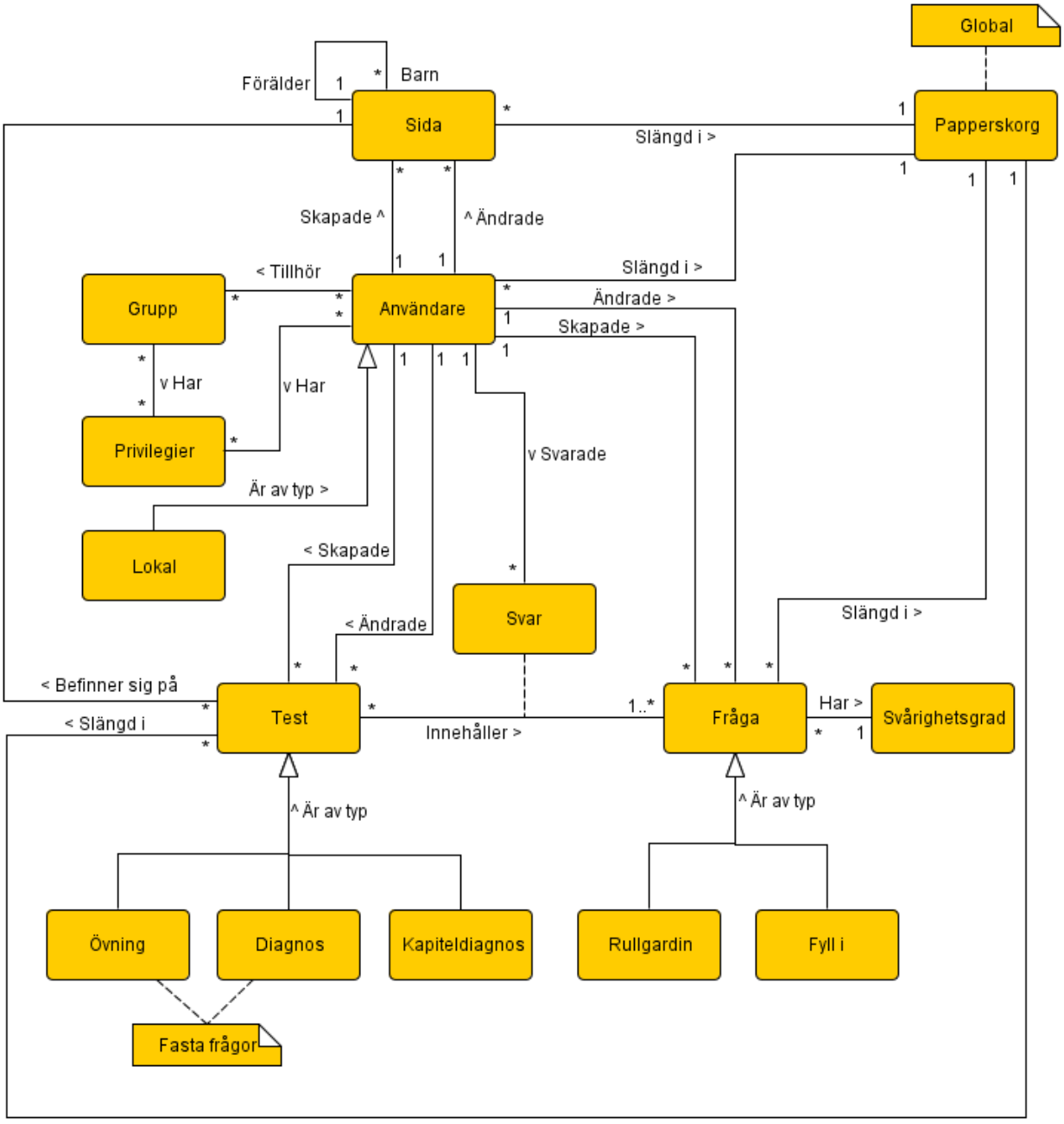
Användarmodellen i databasen behöver vara tillräckligt generell för att kunna hantera användare som inte har ett *Chalmers-id* (CID). Dessutom ställer det nya krav på vilka funktioner och vilken information som ska finnas tillgänglig för administratörer. Lärare på externa platser ska inte ha tillgång till chalmersstudenters resultat och motsvarande för lärare på Chalmers. Systemadministratörer måste fortfarande ha tillgång till hela systemet.

3.1.6 Prestanda och stabilitet

EngOnline har för närvarande vissa stabilitetsproblem när många använder verktyget samtidigt. Det är Fackspråks önskemål att det nya verktyget ska kunna hantera upp till tusen användare. Detta för att tillgodose dagens behov och ett eventuellt ökat behov som skulle tillkomma om fler universitet ska använda verktyget. En simulering av användare behövs för att identifiera prestandabrister och operationer som gör verktyget långsamt. Systemet ska om nödvändigt optimeras så att varje användare använder en skälig del av systemet resurser.

3.2 Domänmodell

Domänmodellen som syns i figur 3.1 visar domänens koncept och deras relationer.



Figur 3.1. Domänmodell.

3.3 Relevanta begrepp för domänen

Begreppsdefinitioner finns i *Appendix C - Vokabulär*.

4. Kravspecifikation

Kraven som ställs på verktyget delas primärt in i funktionella och icke-funktionella krav. Därefter görs en ytterligare uppdelningar inom de funktionella kraven. Dessa delas upp efter vem som använder funktionaliteten och här finns kategorierna *Administratör*, *Student* och *Övrigt*.

4.1 Funktionella krav

De krav som kategoriserats som *Administratör* beskriver den funktionalitet som endast kan utnyttjas av denne. Dessa handlar om funktionalitet så som exempelvis administration av användare, sidor och övningar.

Krav som istället kategoriserats som *Student* beskriver funktionalitet som är riktat till studenterna som använder verktyget, men de kan utföras av alla användare i systemet. Till dessa krav hör till exempel sådan funktionalitet som att genomföra en övning eller att läsa en sida.

Den sista kategorin, *Övrigt*, innefattar krav som inte specifikt tillhör de ovanstående typerna av användare men som istället gäller för verktyget i allmänhet. Ett exempel på detta är kravet på att en extern användare ska kunna identifiera sig med sitt CID.

Samtliga funktionella krav återfinns i *Appendix D - Kravspecifikation*.

4.2 Icke-funktionella krav

De icke-funktionella kraven som beskriver saker såsom tillförlitlighet och underhåll återfinns också i *Appendix D - Kravspecifikation*.

4.3 Användningsfall

Användningsfall återfinns i *Appendix E - Användningsfall*.

Användningsfallens relation till kraven återfinns i *Appendix F - Spårbarhetsmatris*.

5. Analys av möjliga hjälpmedel

Fackspråk ställer inga krav på vilket programmeringsspråk, ramverk eller vilken databashanterare som ska användas. Valet av dessa är ej heller bundet av vad ett specifikt webbhotell kan erbjuda eftersom produkten ska köras på en egenadministrerad server. Lämplig mjukvara kan alltså installeras efter behov. Denna frihet medför att valet av språk och hjälpmedel behöver baseras på de egenskaper som krävs av den färdiga produkten och önskemålet om att de valda hjälpmedlen ska vara fri programvara. Egenskaper som behöver uppfyllas innefattar kravet att användargränssnitt ska vara plattformsoberoende och att verktyget ska vara lätt att underhålla samt enkelt att bygga ut.

5.1 Programmeringsspråk

Att verktyget ska vara lätt att underhålla och bygga ut ställer krav på att det är skrivet i ett språk med en bred användarbas. En mjukvara är inte enkel att underhålla eller att bygga ut om det är svårt att få tag på en utvecklare som kan arbeta i utvecklingsmiljön.

En lista över vilka programmeringsspråk som är populärast över tid, se tabell 5.1, sammanställs av företaget *TIOBE Software*, ett företag som är inriktat på att följa och utvärdera kvaliteten hos mjukvara (TIOBE Software BV, 2013a). Andelen i denna lista baseras på antalet träffar ett språks namn tillsammans med ordet “programming” får hos en mängd olikviktade sökmotorer (TIOBE Software BV, 2013b). Detta kan ge en fingervisning gällande vilka programmeringsspråk som har en aktiv användarbas och används i stor utsträckning.

Tabell 5.1. De tio populäraste programmeringsspråken februari 2013 (TIOBE Software BV, 2013c).

Rank	Programmeringsspråk	Andel
1	Java	18,387 %
2	C	17,080 %
3	Objective-C	9,803 %
4	C++	8,758 %
5	C#	6,680 %
6	PHP	5,074 %

7	Python	4,949 %
8	Visual Basic	4,648 %
9	Perl	2,252 %
10	Ruby	1,752 %

5.2 Plattformsoberoende

Plattformsoberoende för användargränssnittet kan uppnås genom att använda ett programmeringsspråk med egenskapen. Java har denna egenskap och är samtidigt det språk som rankas högst i TIOBEs lista över de populäraste programmeringsspråken. Att använda Java på klientsidan är dock inget alternativ då Fackspråk har uttryckt att detta inte är önskvärt. En anledning är att mindre än en månad tidigare än önskemålet uttrycktes, så upptäcktes ett säkerhetshål som innebar att en underorganisation till amerikanska *Department of Homeland Security* rekommenderade att Java-plugin i webbläsaren skulle inaktiveras (Goodin, 2013; Dormann, 2013). Utöver de säkerhetsbrister som är förknippade med en Java-lösning för klientsidan så innebär det dessutom att alla användare måste installera Java. Detta försämrar tillgängligheten och inför potentiellt fler steg som en användare måste utföra för att använda verktyget. Fler steg innebär en försämrad effektivitet och därmed också en försämrad *användbarhet*, eftersom effektivitet är en grundkomponent i användarbarhet enligt definitionen ISO 9241-11. En försämrad effektivitet kan leda till en försämrad användningsupplevelse (Rogers, Sharp och Preece 2011, ss. 18-19).

En annan lösning för plattformsoberoende användargränssnitt är att använda en webbserver för att tillhandahålla användarna *HTML*, stilmallar och *JavaScript*. Serversidans underliggande teknik blir då inte transparent för klientsidan. Detta görs med hjälp av *HTTP* vars uppgift är att definiera hur en HTTP-klient och HTTP-server kommunicerar (Kurose och Ross 2009, ss. 100-102). För att kunna ta emot data från webbservern så är det enda kravet som ställs på klientsidan att den använder ett program som implementerar klientdelen av HTTP, som en webbläsare.

5.3 Databashanterare och ramverk

Lerner (2010) skriver i den tvåhundrade upplagan av *Linux Journal* att idag är det inget problem att hitta fria varianter av de mjukvaror som krävs för att skapa en webbapplikation. Han framhäver att det är enkelt att hitta operativsystem, databaser, programmeringsspråk och ramverk som fri programvara och menar på att problemet istället ligger i att välja ett av många konkurrerande alternativ som alla har sina egna för- och nackdelar.

5.3.1 Databashanterare

Precis som med övriga mjukvaror i systemet behöver databashanteraren vara etablerad på marknaden med många användare för att den ska vara lätt att underhålla. *Solid IT* har en fortlöpande undersökning av populära databashanterare, där omnämningar på webben samt i jobbannonser ligger till grund för deras poängsats (Solid IT, 2013a). Deras undersökning för databashanterare sammanfattas i tabell. 5.2.

Tabell 5.2: De fyra populäraste relationsdatabashanterare, maj 2013 (Solid IT, 2013b).

Rank	Namn	Poäng
1.	Oracle	1545,86
2.	MySQL	1352,24
3.	Microsoft SQL Server	1352,24
4.	PostgreSQL	182,34

Enligt undersökningen är de tre mest populära databashanterare: *Oracle*, *MySQL* och *Microsoft SQL Server*. Mellan dessa databashanterare är det små skillnader och därefter är det ett större gap ner till den fjärde platsen som innehas av *PostgreSQL*. Solid IT listar också vilken typ varje databashanterare är samt vilka olika stöd de har. De ovan nämnda databashanterare är alla av typen relationsdatabas och stöder främmande nycklar samt *ACID*-transaktioner. Främmande nycklar (Garcia-Molina, Ullman och Widom 2008, s. 311) innebär att värden i ett attribut eller attribut för en relation kräver att det finns motsvarande värde i ett attribut eller som attribut för en annan relation. ACID-transaktioner (Garcia-Molina, Ullman och Widom 2008, s. 9) medför bland annat att om en del av transaktionen misslyckas så ska den ogiltigförklaras och databasen ska återställas till det föregående tillståndet. Dessutom så är Oracle och Microsoft SQL Server kommersiella databashanterare medan MySQL och PostgreSQL har icke-kommersiella alternativ med öppen källkod.

5.3.2 Ramverk för webbutveckling

Lerner (2010) skriver också att modern webbutveckling ofta sker i ramverk. Detta menar han underlättar utvecklingen genom att tillåta ett domänspecifikt fokus istället för återuppfinning av infrastruktur. Han nämner *Ruby on Rails*, *Django*, *Symfony* och *Catalyst* som några av de mer populära ramverken.

5.4 Övervägda ramverk

Det finns många ramverk att välja mellan och de kandidater som övervägdes för användning under utvecklingen var *Spring MVC*, Django, Ruby on Rails och Symfony. Kandidaterna

valdes ut eftersom de bygger på populära programmeringsspråk. Detta avsnitt redogör för stöd och begränsningar hos de ramverk som övervägdes.

5.4.1 Spring MVC

Spring MVC är ett Java-baserat ramverk för webbutveckling. Java är ett plattformsoberorende språk som också kan köras i en webbläsare (Oracle, 2013a). Språket använder ett statiskt typsystem som innebär att alla datatypers kompatibilitet kontrolleras vid kompileringstillfället (Schildt, 2012). Schildt menar att detta kan bidra till att eventuella fel upptäcks tidigare. Ramverket Spring MVC följer designmönstret MVC vilket kommer från engelskans Model, View, Controller. Målet med MVC är att separera gränssnittets och applikationens logik genom att låta kommunikationen mellan dem ske via ett separat lager (Mertic 2009, ss. 11-12). Spring MVC har stöd för en mängd olika databaser av typen *SQL* (Oracle, 2013b) och *NoSQL* (GoPivotal, 2013a). Vidare har ramverket komplett stöd för *AJAX*, *HTML5* och mobila plattformar såsom *Android* och *iPhone* (GoPivotal, 2013b). Spring MVC kan också integreras med sociala medier och har stöd för molnbaserade tjänster (GoPivotal, 2013b).

5.4.2 Django

Django är ett ramverk med öppen källkod för webbutveckling som bygger på programmeringsspråket *Python*. Python är objektorienterat och har en stor mängd standardbibliotek (Python, 2013). Enligt Telles (2008, s. 3) är Python designat för att vara lätt att läsa till exempel genom att använda nyckelord där andra språk använder skiljetecken. Telles menar att detta gör språket enkelt men kan ändå uträtta kraftfulla uppgifter med en liten mängd kod. Django följer designmönstret MVC (Django, 2013a) och har inbyggda system för att autentisera användare och hantera rättigheter för dem (Django, 2013b). Django stödjer databashanterare MySQL, Oracle, *SQLite* och PostgreSQL (Django, 2013c). Django (2013d) fokuserar också på upprepa-inte-dig-själv-principen vilket innebär att man försöker eliminera duplicerad kod så långt som det är möjligt. Mindre mängd duplicerad kod kan i sin tur leda till att systemet blir lättare att underhålla. Webbplatser som *Disqus*, *Instagram* och *Pinterest* använder alla Django (2013d).

5.4.3 Ruby on Rails

Ruby on Rails är ett ramverk för webbutveckling baserat på programmeringsspråket Ruby. Språket Ruby skapades genom att integrera delar från bland annat språk som Perl, Ada och Lisp vilket skapar ett balanserat språk som kombinerar delar av imperativ samt funktionell programmering (Ruby, 2013). Ruby on Rails förespråkar principen om att man inte ska upprepa sig själv (Ruby on Rails, 2013a). Ruby on Rails genererar också kod utifrån vad utvecklaren behöver för att påbörja ett nytt projekt, vilket minimerar den kod som behöver skrivas. Dessutom använder ramverket designmönstret MVC och stöder MySQL, PostgreSQL och har instickskprogram för andra databashanterare (Ruby on Rails, 2013a). Webbplatserna Hulu, Groupon och Github använder Ruby on Rails (2013b).

5.4.4 Symfony

Till språket *PHP* finns ett ramverk för webbutveckling, Symfony. Enligt PHP-manualen (Achour et al., 2013) är PHP ett språk som främst är till för att generera HTML men det kan

också användas som ett generellt skriptspråk eller till att utveckla applikationer. I PHP-manualen står det också att PHP kan användas med procedurell eller objektorienterad programmering. Manualen säger dessutom att språket har stöd för många databashanterare via olika abstraktionslager eller specifika insticksmoduler. Symfony bygger vidare på PHP och har många färdiga funktionaliteter inbyggt. Ramverket är flexibelt då man kan välja vilka delar av det man vill använda och bygga ut med egna moduler (Sensio Labs, 2013a). Dessutom stöder Symfony databaser via *Doctorine*, som till exempel MySQL och PostgreSQL (Sensio Labs, 2013b) för att få en förenklad koppling till relationsdatabaser. Symfony används av bland annat webbplatser såsom *Yahoo!*, *Dailymotion* och *phpBB* (Sensio Labs, 2013c).

5.5 Språk, teknologier, verktyg och hjälpmedel på klientsidan

På klientsidan finns språk, teknologier och verktyg som används för att presentera information i användarens webbläsare. Med hjälp av dessa kan man strukturera och presentera innehållet i en webbapplikation. Webbapplikationen kan även göras mer dynamisk med skriptspråk.

För att strukturera innehåll i en webbapplikation används ofta HTML (Kurose och Ross 2009, s. 100). HTML är ett märkspråk som använder taggar för att strukturera information i ett dokument. Taggarna beskriver vilken typ av struktur eller information som ett avsnitt i dokumentet behandlar. Webbläsare läser de HTML-strukturerade dokumenten som utgör en del av en webbapplikation och visar resultatet för användaren.

För att presentera information från ett dokument i en webbläsare separerat från struktur och innehåll kan man använda märkspråket *CSS* (Cascading Style Sheet). CSS används tillsammans med HTML för att beskriva hur de HTML-definierade taggarna i ett dokument ska presenteras i en webbläsare.

JavaScript är ett skriptspråk som låter utvecklaren skapa dynamiska element i sin webbapplikation. Dynamiska element kan vara delar av ett HTML dokument som döljs eller visas beroende på användarens inmatningar eller rullgardinsmenyer för navigering.

Det finns verktyg som kombinerar eller expanderar de grundstenar som en webbapplikation är uppbyggd av vilket förbättrar utvecklarens möjlighet att skapa system. *Twitter Bootstrap* är ett ramverk som tillhandahåller en fördefinierad design, via CSS och JavaScript, som en utvecklare kan använda sig av för att snabbt skapa ett sammanhängande och användarvänligt gränssnitt. Vidare finns det bibliotek som utökar JavaScript för att underlätta jobbet med att skapa dynamiska webbapplikationer. Ett bibliotek som ger utvecklare sådana möjligheter är *JQuery*.

5.6 Val av ramverk, programmeringsspråk och andra verktyg för produktens realisering

Det ramverk för webbutveckling som valts för att realisera produkten är Django. Detta ramverk bygger på det öppna språket Python, ett av de tio populäraste programmeringsspråken (TIOBE Software BV, 2013a). Pythons popularitet i kombination

med att Djangos officiella hemsida erbjuder gott om dokumentation innebär att det finns hjälpresurser som kan underlätta både den initiala utvecklingen samt underhåll i framtiden. Django har funnits på marknaden i flera år och används av webbplatser som hanterar betydligt fler användare än vad EngOnline behöver kunna göra. Vidare så har det stöd för separation av vy och logik samt stöd för de stora relationsdatabashanterarna. Detta tillsammans med förmågan att leverera ett plattformsoberoende gränssnitt till klienten är vad som krävs för att tillgodose de behov som kravspecifikationen ställer på projektet.

Relationsdatabashanteraren MySQL valdes för att hantera verktygets lagringsrealisering. Som nämnt i avsnitt *5.3.1* är MySQL den mest använda fria databashanteraren på marknaden. Med stöd för främmande nycklar och ACID-transaktioner erbjuder den vad som krävs för att kunna uppfylla kravspecifikationen. Givet den marknadsandel som databashanteraren har så finns det mycket information att hitta och hjälp att få om problem uppstår.

6. Design

En gedigen design underlättar arbetet för alla utvecklare som ska jobba med systemet. För att systemet ska gå att bygga ut och underhållas, även efter det att de ursprungliga utvecklarna inte längre finns tillgängliga, är det viktigt att systemet är väldesignat. Tydlighet i kodstruktur förbättrar systemets läsbarhet för utvecklare. Detta kapitel beskriver hur det nya systemet är konstruerat.

6.1 Arkitektur

För att det ska vara lätt att underhålla systemet är det viktigt att det har en bra arkitektur på alla nivåer. Detta för att man lätt ska kunna sätta sig in i hur systemet fungerar. Mycket av det här arbetet förenklas av att teknologier och verktyg som använts tvingar på utvecklare en bra grund. Det här avsnittet går igenom systemets design från övergripande till mer ingående.

6.1.1 Klient-Server

Systemets mest grundläggande struktur består i en uppdelning av klient och server. Dessa miljöer är skilda åt och kommunicerar endast med hjälp av HTTP via en webbserver. Att serverapplikation bygger på Django och logiken är skriven i Python är ingenting som påverkar klientsidan eftersom den data som skickas med hjälp av HTTP-anrop kommer att vara plattformsoberoende teknik såsom HTML och JavaScript. Processeringen av detta sker sedan i klientens webbläsare.

6.1.2 Uppdelning av funktionalitet

Ett projekt i Django delas upp i vad som kallas *appar* för att gruppera funktionalitet eller koncept som hör ihop. Vidare hanterar appar grundläggande funktionalitet (tilläggning, modifikation och borttagning) för databasen som respektive modell representerar. Systemet är uppdelat i följande appar.

6.1.2.1 Auth

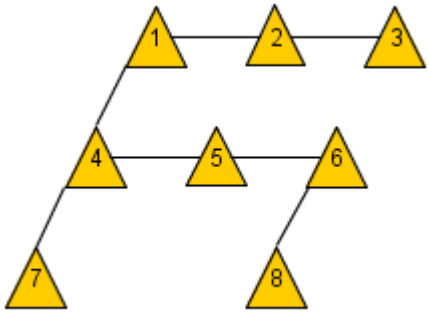
Appen auth hanterar grupper och användare. Appen tillhandahåller funktionalitet för att lista alla systemets användare och grupper. Dessutom finns det stöd för att söka och filtrera listorna. Appen använder sig av Djangos inbyggda autentiseringsapp som innehåller modeller för att representera användare, grupper och deras rättigheter i systemet. I denna app sparas även användarsvar som sammankopplar en användare med ett test och en fråga.

6.1.2.2 Page

Appen page hanterar sidor i systemet. En sida kan innehålla både text och övningar och tillsammans utgör alla sidor en bok. I boken som är indelad i kapitel och underkapitel kan användarna läsa på sidor och utföra övningar.

Sidorna har organiserats i ett vänster-barn-höger-syskon-träd (*VBHS*-träd) för att de ska få en bokstruktur. Fördelen med den här typen av träd är att alla element är ordnade i både höjd- och sidled. Varje element i trädet kan ha ett barn och ett syskon. Syskon ligger på samma nivå medans barn ligger en nivå under. I Fig. 6.1 illustreras ett VBHS-träd där element 1, 2 och 3

är syskon på den högsta nivån. De utgör tillsammans tre kapitel. Elementen 4, 5 och 6 är alla barn till element 1 och bildar tillsammans innehållet i kapitel 1. Element 4 och 6 har både var sitt barn och blir då underkapitel till 1. I appen återfinns också metoder för att utföra operationer på trädet som till exempel insättning och uttagning av element.



Figur 6.1. En datastruktur illustrerat med ett vänster-barn-höger-syskon-träd.

6.1.2.3 Question

Ansvaret för tilläggning, borttagning, förändring och presentation av frågor har denna app. Förutom grundläggande hantering av frågor så sköter appen också rättning av svar som en användare skickar in.

Appen tillåter att frågor har en av två huvudtyper, antingen alternativfråga eller fritextfråga. Alternativfrågor kan ha ett eller flera svar som användare måste välja mellan när denna svarar på dem. Alternativfrågor representeras av rullgardinsmenyer eller kryssrutor beroende på om flera svar är tillåtna eller ej. Fritextfrågor låter däremot användaren skriva in sitt svar som fritext som sedan rättas.

6.1.2.4 Test

Appen test hanterar tilläggning, redigering och borttagning av test. Ett test kan antingen vara en övning, en kapiteldiagnos eller en diagnos.

6.1.2.5 Utils

I appen utils placeras funktionalitet som ska finnas tillgänglig i det övriga systemet. Ofta handlar det om hjälpfunktioner som kan återanvändas i övriga appar för att undvika att kod dupliceras på flera olika ställen enligt den tidigare nämnda principen att inte upprepa sig själv.

6.1.3 MVC-Mönster

Varje app i Django följer internt ett MVC-mönster kallat MTV som innebär att varje app är uppdelat i tre lager: modellen, vyer och mallar (Django 2013a). Varje lager har en tydlig uppgift och ett ansvarsområde vilka förklaras nedan.

Mallarnas uppgift är att producera HTML-kod som ska skickas till klienten. Mallarna skrivs i HTML och ett Django-specifikt mallspråk som processeras på serversidan. Mallspråket

används för att strukturera och presentera data som skickats till mallen. Med mallspråket kan även enklare logik skrivas vilket innebär stöd för if-satser, loopar och variabler. Resultatet blir att mallarna bestämmer innehållet och utseendet för alla sidor i systemet. Den genererade HTML-sidan skickar anrop från klienten till vy hos servern.

Vyernas uppgift är att ta emot och svara på anrop från klienten. Dessa anrop kan handla om att klienten anropar en vy med data som ska manipuleras och valideras för att sedan förändra innehållet i modellen. Alternativt anropas en vy som endast hämtar data från modellen och sedan genererar en HTML-sida av en mall som skickas tillbaka. Med detta har vyerna hand om de logiska delarna av applikationen då det är i vyn som data processeras och HTML genereras från mallar. Vyerna sköter därmed kommunikationen mellan mallarnas genererade HTML och modellen.

Modellen är en abstraktion av databasen. Den består av klasser som är associerade med en tabell i databasen. Varje klass beskriver innehållet i sin tabell. Åtkomst av datan sker genom att skapa instanser av klasserna i modellen. En instans av en klass motsvarar en tabellrad i databasen. Förbindelsen mellan databasen och modellen utgörs av Djangos inbyggda ORM-system (Object Relational Mapping-system).

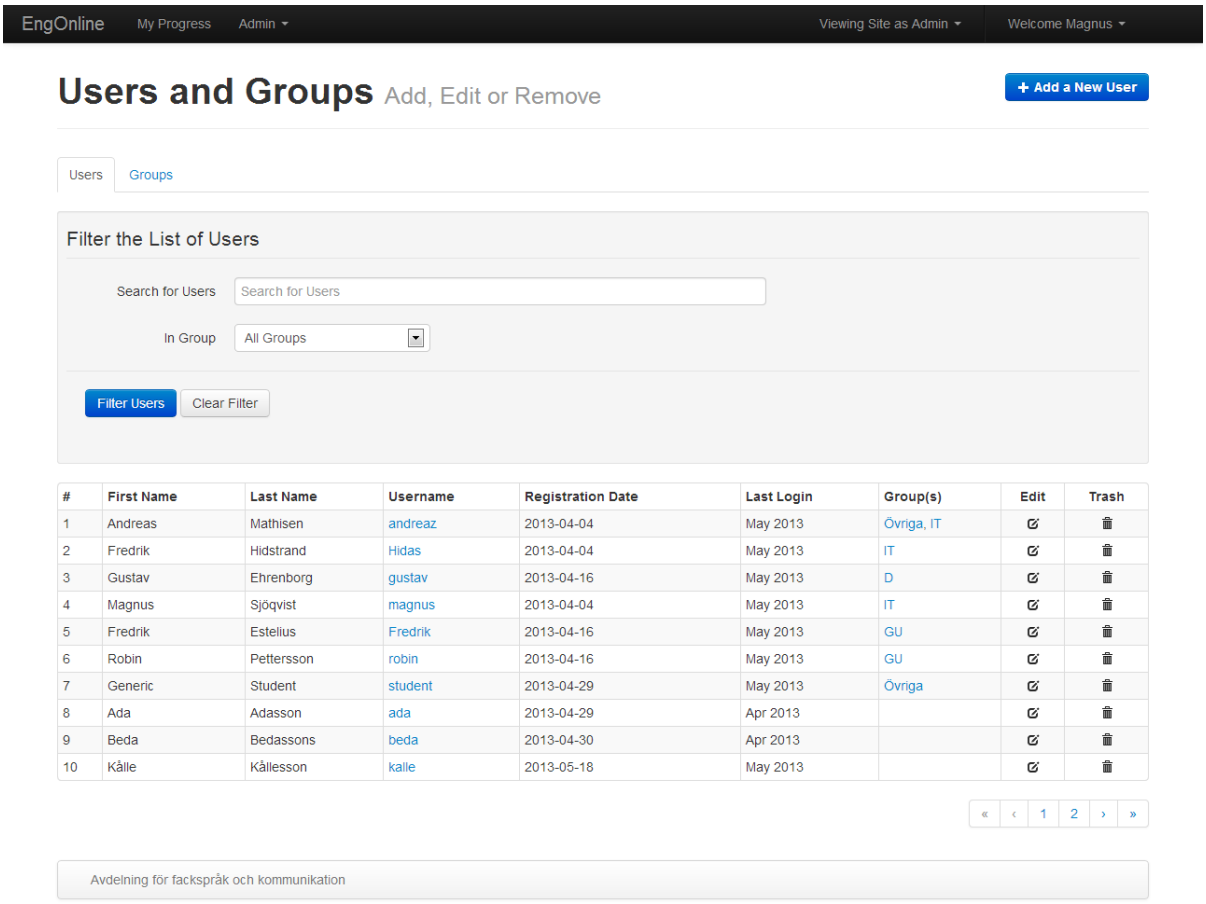
6.1.4 Object Relational Mapping - ORM

För att kommunicera med databasen har Django ett inbyggt ORM-system som gör det möjligt att definiera datamodeller med Python-kod. Via ORM-systemet kan man lagra modellerna i databasen och därefter hämta och spara data via det. Systemet agerar som ett gränssnitt mellan databasen och programmet vilken abstraherar bort databas-kod från programmet. Dessutom skapas objekt som representerar informationen i databasen för att underlätta den kod som skrivs för att manipulera databasen. För att anropa databasen arbetar man då objektorienterat och använder metoder på de genererade objekten. Denna abstraktion gör det trivialt att byta databas då programmet inte behöver skrivas om eftersom ORM-systemet översätter de generella anropen i programmet till den specifika databaskod som den aktuella databasen kräver. Detta innebär att programmet stöder alla databashanterare som Djangos ORM-system stöder.

6.2 Användargränssnittet

Användargränssnittet ska vara användarvänligt och modernt. Det senare har gjorts med hjälp av Twitter Bootstrap, ett bibliotek för utseende som erbjuder färdiga vy-komponenter såsom knappar, rullgardinsmenyer och innehållsstrukturering. Det konsekventa användandet av biblioteket har medfört att verktyget har ett enhetligt utseende. Ett enhetligt utseende där användaren kan känna igen sig och där liknande funktionalitet ser likadan ut gör att gränssnittet blir mer användarvänligt (Rogers, Sharp och Preece 2011, s. 506). Ett exempel på detta är att listning och filtrering av frågor, användare och grupper har ett konsekvent utseende. Ytterligare ett exempel är en konsekvent användning av knappar där ett kryss alltid är borttagning och en papperskorg alltid innebär en borttagning som går att ångra.

Överallt i verktyget finns alltid en global navigering tillgänglig, denna menyrad syns överst i figur 6.1. Med hjälp av denna kan studenten alltid nå sidan för att se sin kunskapsutveckling och administratören kan dessutom nå administrationsmenyn. Administratören kan också komma åt funktionen för att växla mellan att titta på sidan som administratör eller som student.

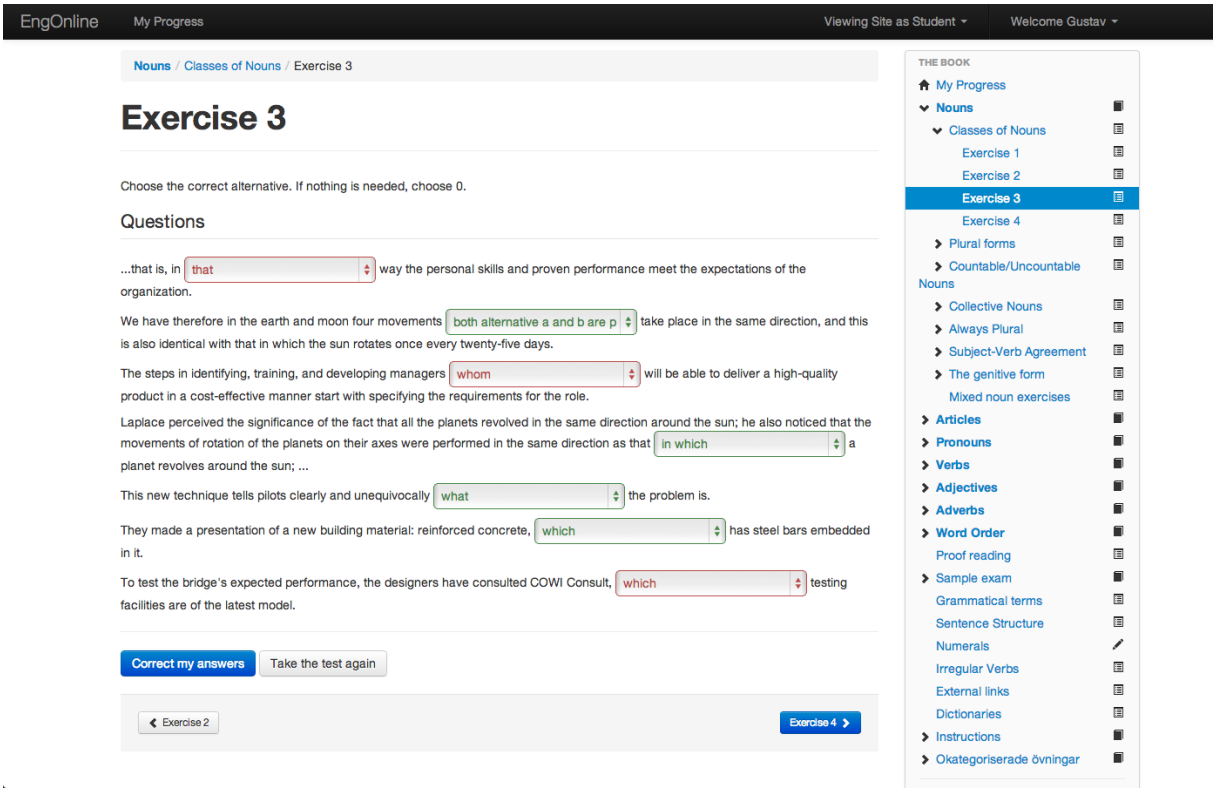


Figur 6.1. Vyn för att lista och söka efter användare.

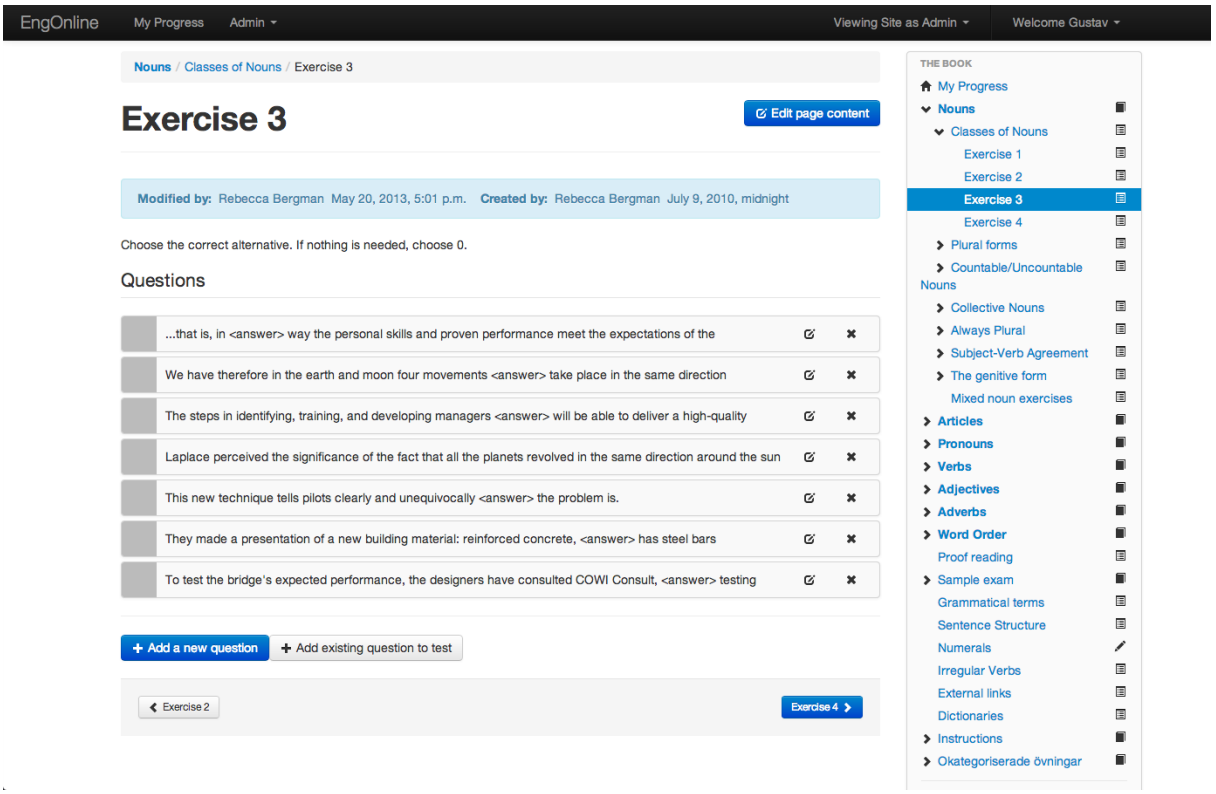
Funktionen som gör att administratören kan se verktyget ur studentens perspektiv har gjort det möjligt att specialisera de båda vyerna. Detta innebär att de kan få ett tydligare fokus och ett färre antal komponenter. För många visuella komponenter kan enligt *Nielsens Hueristics* leda till en sämre användarvänlighet eftersom de försvinner i mängden (Rogers, Sharp och Preece 2011, s. 507).

Att studentvyn och administratörsvyn skiljer sig åt blir tydligt när man visar en övning. För en student har en övning en rubrik, eventuellt en inledande text följt av övningens frågor. Fyller man i frågorna och trycker på knappen för att rätta så kommer resultatet tillbaka i form av färgkodad rättning. Detta syns i figur 6.2. För administratören försvinner funktionaliteten för att svara och rätta frågor och istället finns administratörskomponenter. I figur 6.3 visas samma övning som i figur 6.2 men i administratörsläget. Här finns möjlighet att dra och släppa

frågorna för att ändra ordning. Det är också möjligt att ta bort och lägga till frågor samt att ändra övningens text eller frågorna.



Figur 6.2. En övning i studentvyn. Användare kan navigera med hjälp av framåt- och bakåtknapparna som finns under frågorna eller med trädnavigeringen till höger.



Figur 6.3. En övning i adminstratörsläget. Gränssnitt för att genomföra övningen har ersatts med ett som tillåter administrering.

6.3 Databasmodell

ER-diagram över databasmodellen har härletts från det abstrakta domänmodell-diagrammet. Diagrammet återfinns i *Appendix G - Entity-Relationship diagram*.

7. Resultat

Projektet resulterade i en produkt som uppfyller majoriteten av de specificerade kraven. Detta kapitel beskriver vilka krav som uppfylldes, resultatet av databasmigreringen, några av de problem som stöttes på under utvecklingens gång och intressanta kodlösningar.

7.1 Uppfyllande av krav

Krav av prioritet ett och två är i stort uppfyllda. De krav som inte uppfyllts handlar om funktionalitet som berör externa användare. Varför dessa krav inte uppfyllts beskrivs i avsnittet *7.3 Problem*.

Flertalet krav av prioritet tre har också uppfyllts. Den fullständiga kravspecifikationen och vilka av kraven som uppfyllts finns i *Apppendix E - Krav*.

7.2 Migrerad data

Ett av målen med projektet var att det nya verktyget skulle återanvända innehållet från det gamla EngOnline. Alla sidor och övningar har migrerats till det nya systemet och över 90 % av alla frågor. De frågor som ännu inte har migrerats är av två typer som ännu inte är kompatibla med det nya systemet.

7.3 Problem

Under arbetets gång var det ett antal större problem som behövde lösas. Dessa problem handlade om migreringen av databasen, inloggning med hjälp av CID, polymorfism och programstruktur.

7.3.1 Migrering av databasen

Under migreringen av databasen uppstod en stor mängd problem. Designen av den gamla databasen skiljde sig nämnvärt från hur den nya databasen skulle se ut. Detta fick som följd att hela databasmigreringen inte kunde automatiseras på ett enkelt sätt utan var tvungen att delas upp i flera steg varav ett par var manuella.

Databasen innehöll också väldigt mycket data som inte hörde till EngOnline. Detta försvårade arbetet med att skaffa en överblick över systemet i sin helhet. Kolumnerna i databasen hade inte heller alltid tydliga eller relevanta namn vilket gjorde det svårt att läsa ut vad det var för data som sparats i tabellerna. Dessa skäl gjorde det svårt att hantera databasen och ökade komplexiteten för hela databasmigreringen.

Datan som lagrats i databasen var ofta lagrad med både logisk struktur och utseendespecifikation. Då HTML-koden i det nya systemet är skrivet på ett sätt där den logiska strukturen och utseendespecifikationen skiljs åt från varandra för att enklare kunna administreras skapas det konflikter med den befintliga HTML-koden som redan finns i datan från databasen.

7.3.2 Externa användare

Ett av kraven på systemet var att studenter på Chalmers ska kunna logga in med sina CID. Det kravet uppfylls inte av produkten i dagens läge. Denna funktionalitet skulle gå att implementera med protokollet *SAML2* enligt Chalmers IT-service (ITS). För att det ska gå att kommunicera med SAML2 krävs det dock att Chalmers servrar konfigureras för att fungera med det nya systemet. På grund av brister i kommunikationen med ITS har varken Chalmers servrar eller det nya systemet konfigurerats.

7.3.3 Djangos ORM-systems hantering av arv

Klasser i Python kan ärva sin information och funktionalitet från andra klasser och det reflekteras i Django då arv finns tillgängligt för modeller. Detta fungerar dock inte som väntat i kombination med Djangos ORM-system.

Det uppstår problem vid användandet av överskuggade metoder i subklasser. Med polymorfism så ska den överskuggade metoden i klassen för objektets egentliga typ utföras. Men på grund av Djangos ORM-system så anropas istället den metod som återfinns i klassen för objektets nuvarande typ vilket bryter mot detta.

Istället för polymorfism använder Django sig av principen att anta typen på objekt och hantera då fel inträffar. Detta kommer från Python och motiveras av att rätt typ kan antas och fel inträffar endast i specialfall som då är enkla att hantera.

7.4 Hantering av polymorfism i Django

För att komma runt problemet med bristande stöd för polymorfism i Django så finns en metod i varje klass som returnerar klassens typ. Anropet av dessa metoder kommer alltid att anropa metoden i objektets aktuella klass. Där testas först om objektet är av subklassernas typ och om så är fallet skickas anropet rekursivt till denna subklass. Detta fortgår tills ingen mer subklass matchar objektets verkliga typ och då returneras den klass som definierar subklassen. Det samlar all kod som gör antagande om objektens typ i objektens klasser som annars skulle spridas över hela applikationen där objekts typ testas. Den stora nackdelen med denna lösning är att de rekursiva anropen måste göras explicita så att man måste för varje subklass skriva klassens namn i anropet. Detta innebär att vid tillägg av en subklass behöver metoder skrivas om högre upp i klasshierarkin för att kunna testa typen hos subklassen.

8. Diskussion

Majoriteten av det kravställda arbetet genomfördes inom tidsramen. Viktiga punkter som val av hjälpmedel, vad som gick bra, vad som kunde förbättras och vad som kunde gjorts annorlunda diskuteras i det här kapitlet.

8.1 Valet av hjälpmedel

I analysen av möjliga hjälpmedel skrevs att det finns många alternativ att välja mellan när det gäller ramverk för webbutveckling. De olika ramverken har fördelar och nackdelar som kan vägas mot varandra men i stort erbjuder de likvärdig funktionalitet relaterat till de krav som ställts på projektet. Detta eftersom samtliga ramverk som övervägdes har stöd för separation av logik och vy. Vidare har alla stöd för flertalet populära databashanterare. Dessutom bygger alla på något av de tio populäraste programmeringsspråken, vilket bör innebära att det finns gott med tillgängliga hjälpresurser.

Valet får dock betydelse eftersom ramverket i sig och det programmeringsspråk som då medföljer sätter sin prägel på både stil och struktur. Detta var en av de faktorer som blev avgörande för valet av Django. En annan anledning till valet av Django var dess inbyggda funktionalitet för autentisering av användare som överensstämde med domänmodellen.

8.2 Positiva aspekter av projektet

De grundläggande kraven implementerades, kommunikationen med kunden fungerade bra och den agila arbetsprocessen var effektiv.

8.2.1 Implementering av grundläggande krav

Av de krav som ställdes på verktyget blev majoriteten av de högst prioriterade implementerade. Dessutom implementerades många av de lägst prioriterade kraven som endast skulle implementeras i mån av tid.

8.2.2 Kommunikation med kunden

Kommunikationen med kunden fungerade bra. Gemensamma möten gjorde att kunden hölls uppdaterad om projektets status samtidigt som gruppen kunde ställa frågor. Kommunikationen gjorde att kunden och gruppen tillsammans kunde reda ut oklarheter.

8.2.3 Den agila arbetsprocessen

Genom att arbeta agilt underlättades utvecklingen av systemet. Den agila arbetsprocessen, som tidigare beskrivits, gjorde att utvecklare snabbt kunde ta hjälp av varandra för att lösa problem som uppstod under arbetet. Genom att inte behöva skriva minutiösa specifikationer för hur de individuella delarna av systemet skulle kopplas samman kunde tid istället läggas på att vidareutveckla systemet med stöd av direktkommunikationen mellan utvecklarna. Frågor

kring en del av systemet kunde snabbt riktas till den utvecklare som hade det primära ansvaret för den, vilket gjorde att utvecklare snabbt kunde lösa problem som uppstod i relation till den delen av systemet.

8.3 Vad som kunde gjorts bättre

För projektet i sin helhet finns det utrymme för förbättring gällande kommunikation med utomstående och testning av koden.

8.3.1 Kommunikation med Chalmers IT-support

Kommunikationen med ITS gällande autentisering mot Chalmers inloggningssystem borde ha inletts tidigare i utvecklingsfasen. Kommunikationen tog längre tid än förväntat vilket i slutändan ledde till att funktionaliteten inte kunde implementeras. Om kommunikationen initierats tidigare kanske funktionaliteten hade hunnits implementeras.

8.3.2 Testning av koden

Det saknas oftast tester för metoder vilket kan göra det svårare att upptäcka fel som uppstår vid ändring av kod. Tester kan dessutom fungera som en form av dokumentation för koden eftersom de specificerar vad som bör och inte bör tillåtas av koden.

8.4 Alternativa tillvägagångssätt

För att framtida system ska bli bättre är det viktigt att identifiera vad som kunde ha gjorts på ett bättre sätt. För projektet hade det varit bättre att göra en grundligare studie av ramverket, designat systemet med utgångspunkt i ramverket och prioriterat krav annorlunda.

8.4.1 Grundligare studie av Django

En grundligare studie av Django skulle underlätta implementeringen av projektet. För att kunna strukturera systemet på ett bra sätt är det fördelaktigt om utvecklarna har goda kunskaper om det verktyg som ska användas. Genom att använda ett ramverk som Django får utvecklarna mycket stöd i utvecklingen. För att på ett bra sätt ta vara på stödet från ramverket är det bra om utvecklarna har goda kunskaper om det. Samtidigt så gäller det att hitta en balans mellan att studera teori och att få praktiska erfarenheter.

8.4.2 Implementering av databasen

Databasen implementerades först med SQL, därefter genererades django-modeller utifrån den. De genererade modellerna representerade inte databasen på ett korrekt sätt. Det ledde till att det i efterhand behövde ändras i både modellerna och databasen. Om modellerna istället hade definierats i Python från början och låtit databasen genereras från dem hade den problematiken inte behövt uppstå.

8.4.3 Strukturering av projektet i appar

I Django struktureras projekt med hjälp av appar. Som tidigare nämnts är ambitionen att en app hanterar funktionalitet för en del av systemet. Problemet är att sköta uppdelningen i praktiken.

Ett tänkbart sätt att dela upp projektet skulle vara i en admin app och en student app. Admin appen skulle då ansvara för att hantera all administrering så som att lägga till, ta bort eller ändra information i databasen. Appen för studenter sköter studentdelen av projektet, det vill säga att utföra övningar, etcetera. Nackdelen med den här uppdelningen är att hanteringen av till exempel frågor i systemet delas upp i de två olika apparna. Att visa och rätta frågor hamnar i student-appen medans hanteringen av frågor i databasen sköts av admin-appen. Allt som rör frågor hanteras därför inte på ett och samma ställe.

En annan lösning är att dela upp apparna utifrån funktionalitet. All funktionalitet som rör frågor hamnar i appen för frågor, medans all funktionalitet som rör sidor hamnar i appen för sidor, och så vidare. Att dela upp projektet på det här viset gör att varje app sköter en del av projektet, men alla apparna kan bli beroende av varandra för att tillsammans utgöra projektet. Det är alltså inte säkert att en app är så fristående från övriga appar i systemet som strukturen låter påskina.

8.4.4 Alternativ metod och prioritering av krav

I projektet prioriterades krav som generellt har med att skapa och redigera innehåll i systemet högt, men eftersom allt innehåll till EngOnline redan existerade när projektet började var behovet av att skapa nytt innehåll litet. Fokus borde kanske därför ha legat på funktionalitet som gör det gamla innehållet tillgängligt för studenterna och därefter funktionalitet för att skapa nytt innehåll. Det hade gjort att system hade kunnat börjat användas av studenterna i ett tidigare skede som i sin tur hade kunnat resultera i nyttig feedback.

8.5 Sammanfattande kommentar

Fackspråk önskade ett nytt system som skulle ersätta det gamla EngOnline. Det nya verktyget för undervisning i engelsk grammatik skulle till skillnad från det gamla vara väldokumenterat, utbyggbart och användarvänligt. Krav ställdes och prioriterades i samråd med Fackspråk. Majoriteten av kraven prioriterades som essentiella medans resterande var ämnade för framtiden eller implementering i mån av tid. Det nya verktyget uppfyller inte bara de krav som prioriterades som fundamentala, utan också en del av de övriga.

Verktyget realiserades med hjälp av en klient-server-lösning där Django, ett ramverk för webbutveckling implementerat i Python, användes på serversidan. Tillsammans med en webbserver förser detta klientsidan med HTML, stilmallar och skript. Django lämpade sig tack vare stöd för MVC-mönstret och ett bibliotek för autentisering som överensstämde med domänmodellen. Dessutom kan databasmodell och interaktion med denna göras icke-implementationspecifik med hjälp av det inbyggda ORM-systemet.

Att använda ett ramverk för webbutveckling innebär att en fördefinierad infrastruktur används. Detta medför att mindre mängd kod behöver skrivas och att koden som skrivs generellt blir mer domänspecifik. Därför borde det leda till att det blir enklare att sätta sig in i projektet, speciellt om man känner till ramverket sen tidigare. Detta tillsammans med separation av vy och logik samt verktygets dokumentation bör ge en god grund för vidareutveckling.

Det nya systemets utseende är implementerat med hjälp av Twitter Bootstrap. Detta har lett till ett modernt och enhetligt gränssnitt. Gränssnittets användarvänlighet bygger på följda designprinciper och den feedback som regelbundna möten med kunden inneburit.

Det finns ett antal områden som skulle innebära en naturlig fortsättning på projektet. Bland annat att genomföra användartester för att utvärdera gränssnittet. Dessutom behövs den externa autentiseringen via Chalmers implementeras. Det saknas också stöd för vissa typer av frågor som då ännu inte kan föras över från det gamla EngOnline. Ett annat område för vidareutveckling i framtiden är Fackspråks önskan om att i framtiden kunna ladda upp och inkludera media i verktyget. Utöver detta så finns det önskemål om att kunna ta fram och visa relevant statistik för användarna.

Dessutom finns aspekten pedagogik. Här skulle verktyget kunna utvecklas med exempelvis nya typer av övningar eller förbättring av gränssnittet hos de som redan existerar.

Källförteckning

Achour, M. et al. (2013) PHP Manual. http://www.php.net/manual/en/intro-whatcando.php (2013-05-07)

Cockburn, A. och Highsmith, J. (2001). Agile software development, the people factor. *Computer*, vol. 34, nr 11, ss. 131-133.

Django (2013a) FAQ: General. *Django*. https://docs.djangoproject.com/en/dev/faq/general (2013-05-09)

Django (2013b) User authentication in Django. *Django*. https://docs.djangoproject.com/en/1.5/topics/auth/ (2013-05-09)

Django (2013c) Databases. *Django*. https://docs.djangoproject.com/en/1.5/ref/databases/ (2013-05-09)

Django (2013d) Meet Django. *Django*. https://www.djangoproject.com/ (2013-05-09)

Dormann, W. (2013) Vulnerability Note VU#625617 - Java 7 fails to restrict access to privileged code. *Vurnability Notes Database*. http://www.kb.cert.org/vuls/id/625617 (2013-03-02).

Garcia-Molina, H., Ullman, J. D., och Widom, J. (2008) *Database Systems: The Complete Book*. Upplaga 2. Upper Saddle River: Prentice Hall Press.

Goodin, D. (2013) Critical Java vulnerability made possible by earlier incomplete patch (Updated). Ars Technica. 11 januari. http://arstechnica.com/security/2013/01/critical-java-vulnerability-made-possible-by-earlier-incomplete-patch/ (2013-03-02).

GoPivotal (2013a) DATA ACCESS. *Spring*. http://www.springsource.org/features/data-access (2013-05-20)

GoPivotal (2013b) FEATURE TOUR. *Spring*. http://www.springsource.org/features (2013-05-14)

Kurose, J. och Ross, K. (2009) *Computer Networking: A Top-Down Approach*. Upplaga 5. Boston: Pearson Education, Inc.

Larman, C. (2004) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Upplaga 3. Boston: Addison-Wesley.

Lerner, R. M. (2010) At the Forge. *Linux Journal*, vol. 2010 nr. 200, s. 4.

Mertic, J. (2009) MVC Architecture. I *The Definitive Guide to SugarCRM*, ss. 11-12. Apress.

Oracle (2013a) Learn About Java Technology. *Java*. http://www.java.com/en/about/ (2013-05-14)

Oracle (2013b) Java SE Technologies - Database. *Oracle*. http://www.oracle.com/technetwork/java/javase/jdbc/index.html (2013-05-14)

Python (2013) About Python. *Python*. http://www.python.org/about/ (2013-05-10)

Rogers, Y., Sharp, H. och Preece, J. (2011) *Interaction Design: Beyond Human-Computer Interaction*. Upplaga 3. Chichester: John Wiley & Sons.

Ruby (2013) About Ruby. http://www.ruby-lang.org/en/about/ (2013-05-08)

Ruby on Rails (2013a) Getting Started with Rails. *RailsGuides.* http://guides.rubyonrails.org/getting_started.html (2013-05-09)

Ruby on Rails (2013b) Real applications live in the wild. *Ruby on Rails: Applications*. http://rubyonrails.org/applications (2013-05-09)

Schildt, H. (2012) Java: a beginner's guide, fifth edition. *Books24x7.* http://common.books24x7.com.proxy.lib.chalmers.se/toc.aspx?bookid=44138 (2013-06-06)

Sensio Labs (2013a) The technological benefits of Symfony in 6 easy lessons. http://symfony.com/six-good-technical-reasons (2013-05-07)

Sensio Labs (2013b) Databases and Doctrine http://symfony.com/doc/2.1/book/doctrine.html (2013-05-08)

Sensio Labs (2013c) About. http://symfony.com/about (2013-05-07)

Solid IT (2013a) Method of calculating the scores of the DB-Engines Ranking. *DB-Engines.* http://db-engines.com/en/ranking_definition (2013-05-20)

Solid IT (2013b) DB-Engines Ranking. *DB-Engines.* http://db-engines.com/en/ranking/relational+dbms (2013-05-02).

Telles, M. A. (2008) About Python. I *Python power!: the comprehensive guide*, s. 3. Course Technology.

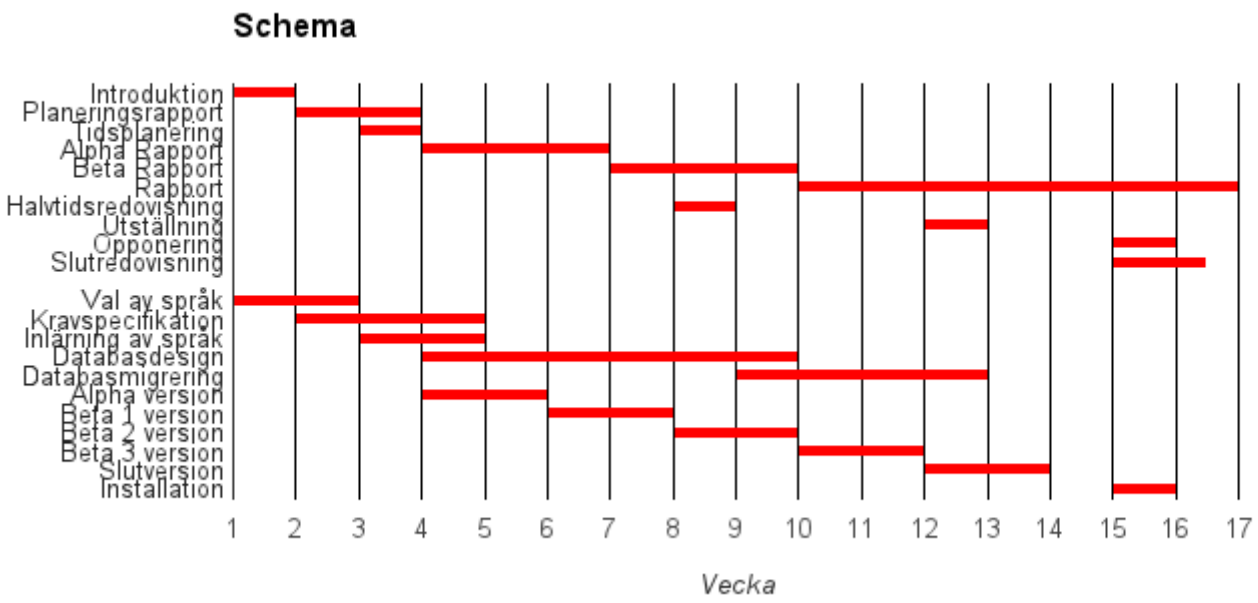
TIOBE Software BV (2013a) *TIOBE Software: General Information*. http://www.tiobe.com/index.php/content/company/GeneralInfo.html (2013-03-02).

TIOBE Software BV (2013b) *TIOBE Index Definition*. http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm (2013-03-02).

TIOBE Software BV (2013c) *TIOBE Index*. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (2013-03-02).

Appendix A - Tidsplan

Gantt-schemat nedan representerar en grov tidsplanering.



Datum/veckor för delrapporter samt slutgiltig presentation:

2013-02-15 - Inlämning av planeringsrapport

2013-03-19 - Halvtidsredovisning av projektgrupperna

2013-05-20 - Inlämning av slutrapport

2013-05-21 - Utställning

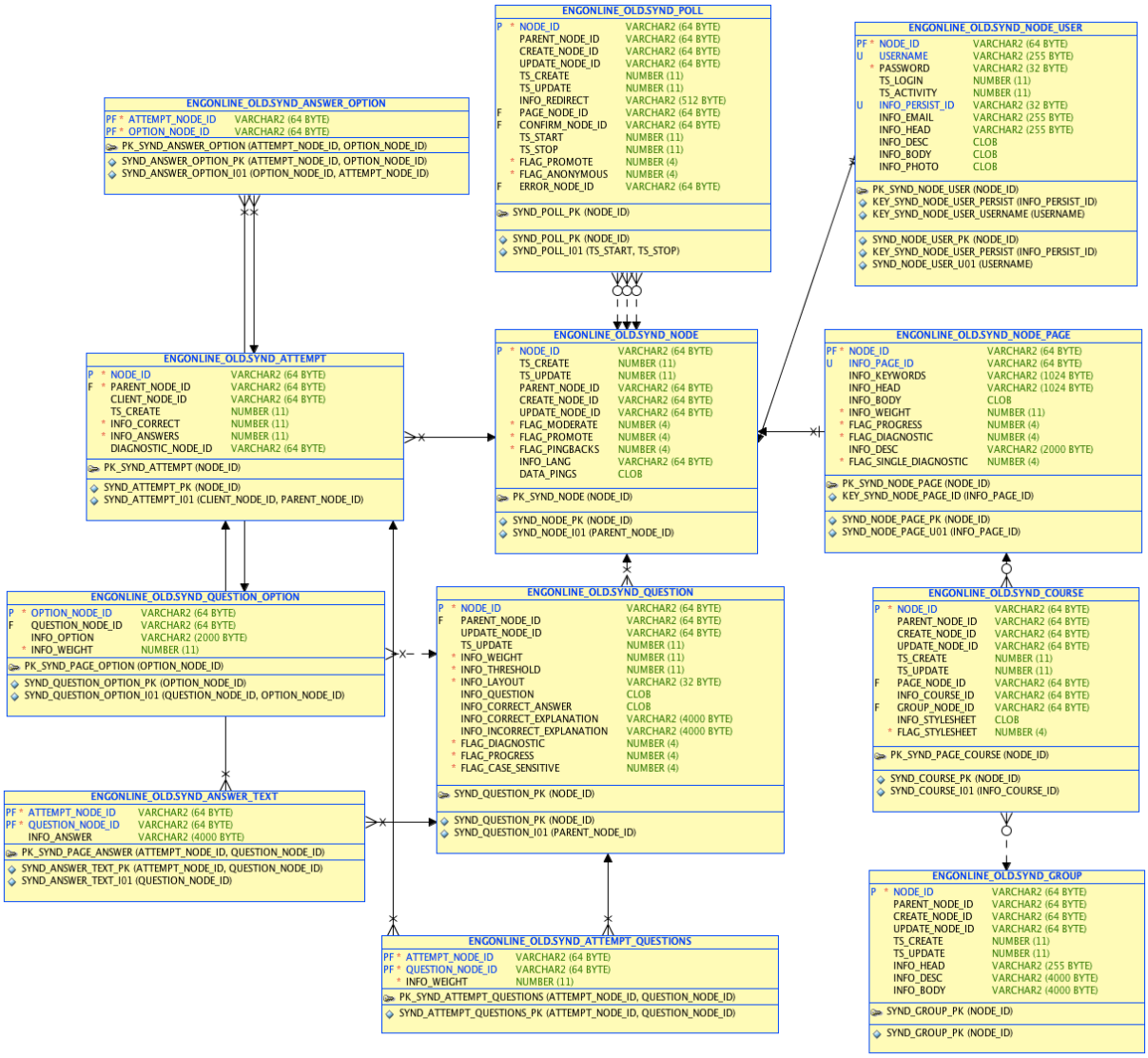
2013-05-29 - Inlämning av skriftlig opposition

2013-06-03 - Muntlig slutredovisning, dag 1

2013-06-04 - Muntlig slutredovisning, dag 2

2013-06-07 - Inlämning av slutrapport med opponentkommentarer.

Appendix B - Entity-relationship diagram för den gamla databasen



Appendix C – Vokabulär

Administratör:

En användare som har rättigheter att hantera sidor, användare, övningar, progress checks och diagnoser. En admin kan lägga till eller ta bort administratör-status för andra användare. En administratör har dessutom alla rättigheter som student har.

Användare:

Alla konton i systemet räknas som användare.

Diagnos:

Ett test en användare kan genomföra för att testa sina kunskaper på hela grammatikkursens innehåll.

EngOnline:

Namnet på språkverktyget.

E-R diagram

Entity–relationship model diagram - en diagramtyp som används för att beskriva databaser.

Extern användare:

En användare som autentiserar sig mot en extern server. En extern användare som tas bort ur systemet kommer fortfarande att kunna logga in men all data från tidigare användning kommer att vara borta.

Inloggningsuppgifter:

Användarnamn och lösenord.

Lokal användare:

En användare som autentiserar sig mot det lokala systemet. En lokalanvändare som tas bort ur systemet kommer inte att kunna logga in igen till skillnad från en externa användare.

Kapiteldiagnos:

Ett test en användare kan genomföra för att testa sina kunskaper på ett enskilt kapitel.

Sida:

En nod i språkverktygets trädstruktur. Kan innehålla text, övningar, progress checks och diagnoser.

Student:

En användare som har rättigheter att läsa sidor och att göra övningar, progress checks och diagnoser.

Appendix D – Kravspecification

Krav för administratörer:

Id	Namn	Prioritet	Uppfyllt
1.1	En administratör ska kunna lägga till lokala användare.	2	OK
1.2	En administratör ska kunna ändra lokala användare.	3	OK
1.3	En administratör ska kunna ta bort lokala användare.	2	OK
1.4	En administratör ska kunna lägga till externa användare.	2	
1.5	En administratör ska kunna ändra externa användare.	3	
1.6	En administratör ska kunna ta bort externa användare.	2	
1.7	En administratör ska kunna återställa borttagna användare.	3	OK
1.8	Systemet ska stödja användarklasser.	2	OK
1.9	En administratör ska kunna redigera rättigheter för användare.	2	OK
1.10	En administratör ska kunna söka efter specifika användare.	2	OK
1.11	En administratör ska kunna lägga till artiklar.	1	OK
1.12	En administratör ska kunna ändra artiklar.	1	OK
1.13	En administratör ska kunna ta bort artiklar.	1	OK
1.14	En administratör ska kunna lägga till övningar.	1	OK
1.15	En administratör ska kunna ändra övningar.	2	OK
1.16	En administratör ska kunna ta bort övningar.	1	OK

1.17	En administratör ska kunna lägga till diagnoser.	1	OK
1.18	En administratör ska kunna ändra diagnoser.	2	OK
1.19	En administratör ska kunna ta bort diagnoser.	1	OK
1.20	En administratör ska kunna lägga till frågor.	1	OK
1.21	En administratör ska kunna ändra frågor.	3	OK
1.22	En administratör ska kunna ta bort frågor.	1	OK
1.23	En administratör ska kunna återställa borttagna artiklar/frågor/diagnoser.	2	OK
1.24	En administratör ska kunna flytta frågor inom övningar.	2	OK
1.25	En administratör ska kunna flytta frågor inom diagnoser.	2	OK
1.26	En administratör ska kunna flytta frågor mellan övningar.	2	OK
1.27	En administratör ska kunna markera en fråga för att vara med i kapiteldiagnoser.	3	
1.28	En administratör ska kunna se alla frågor som är tillagda i kapiteldiagnoser.	3	
1.29	En administratör ska kunna lägga till lokala användare i “batch”-stil.	3	
1.30	En administratör ska kunna lägga till en grupp.	3	OK
1.31	En administratör ska kunna ändra en grupp.	3	OK
1.32	En administratör ska kunna ta bort en grupp.	3	OK
1.33	En administratör ska kunna lägga till en kapiteldiagnos.	2	OK

1.34	En administratör ska kunna ta bort en kapiteldiagnos.	2	OK
1.35	En administratör ska kunna ändra artiklarnas trädstruktur.	2	OK
1.36	En administratör ska kunna se användare	2	OK
1.37	En administratör ska kunna se sidor som en student	3	OK

Krav för studenter:

Id	**Namn**	**Prioritet**	**Uppfyllt**
2.1	En student ska kunna se tillgängliga artiklar/övningar/diagnoser.	1	OK
2.2	En student ska kunna navigera till tillgängliga artiklar/övningar/kapiteldiagnoser/diagnoser.	1	OK
2.3	En student ska kunna se artiklar/övningar/kapiteldiagnoser/diagnoser	1	OK
2.4	En student ska kunna genomföra övningar.	1	OK
2.5	En student ska kunna genomföra kapiteldiagnoser.	2	OK
2.6	En student ska kunna genomföra en diagnos.	1	OK
2.7	En students resultat från en diagnos ska sparas.	1	OK
2.8	En students resultat från en kapiteldiagnos ska sparas.	2	OK
2.9	En students resultat från en övning ska gå att spara.	1	OK
2.10	En student ska kunna se statistik över sina resultat.	3	

2.11	En student ska kunna se gamla resultat.	3	OK
2.12	En student ska kunna se vilken kategori en besvarad fråga tillhörde.	3	

Övriga krav:

Id	Namn	Prioritet	Uppfyllt
3.1	En extern användare ska kunna autentisiera sig med sitt CID.	1	
3.2	En lokal användare ska kunna autentisiera sig.	1	OK
3.3	En genomförd övning ska ge färgkodad feedback.	1	OK
3.4	En användare ska kunna tillhöra en grupp.	2	OK
3.5	En användare eller sida som tas bort ska markeras som borttagen men fortfarande finnas kvar i databasen.	2	OK
3.6	En fråga ska kunna markeras med olika svårighetsgrader.	1	OK
3.7	En fråga ska kunna markeras som “på svenska” eller “på engelska”.	1	OK
3.8	En fråga ska kunna tillhöra en kategori.	1	OK
3.9	En engelsk kapiteldiagnos ska inte innehålla frågor som markerats som “på svenska”.	2	
3.10	En sida ska kunna innehålla videoklipp	3	
3.11	En sida ska kunna innehålla länkar till andra sidor	1	OK
3.12	En användare ska kunna logga ut	1	OK

Icke-funktionella krav:

Id	Namn
4.1	Systemet ska utnyttja MVC-mönstret.
4.2	Systemet ska inte vara beroende av en specifik databasimplementation.
4.3	Databasen ska använda ACID-transaktioner.
4.4	Lokala användares lösenord ska lagras krypterat.
4.5	URLen ska inte avslöja underliggande teknik.
4.6	Användargränssnittet ska vara plattformsoberoende

Appendix E - Användningsfall

Administratör:

Id: **1.1**

Namn: Lägg till en lokal användare

Prioritet: 2

Aktörer: Administratör

Mål: Lägga till en lokal användare i systemet

Beskrivning: Administratören väljer att lägga till en lokal användare i systemet. Administratören fyller i användaruppgifter och eventuell grupp och väljer sedan att spara. Systemet lägger till den nya användaren.

Id: **1.2**

Namn: Redigera en lokal användare

Prioritet: 3

Aktörer: Administratör

Mål: Redigera en lokal användare i systemet

Beskrivning: Administratören väljer att redigera en användare i systemet. Administratören redigera aktuella användaruppgifter och eventuell grupp och väljer sedan att spara. Systemet sparar de ny uppgifterna.

Id: **1.3**

Namn: Ta bort en användare

Prioritet: 2

Aktörer: Administratör

Mål: Ta bort en användare ur systemet

Beskrivning: Administratören väljer att ta bort en användare ur systemet. Användare tas bort ur systemet.

Id: **1.4**

Namn: Redigera rättigheter för en användare

Prioritet: 2

Aktörer: Administratör

Mål: Ändra rättigheterna för en användare

Beskrivning: Administratören väljer att redigera rättigheterna för en användare. Rättigheterna för användaren ändras om adminstratören har rättigheter att genomföra ändringen.

Id: **1.5**

Namn: Lägga till en sida

Prioritet: 1

Aktörer: Administratör

Mål: Lägga till en sida i systemet

Beskrivning:	Administratören väljer att lägga till en sida i systemet. Administratören fyller i informationen som ska synas på sidan och väljer sedan spara. Systemet sparar och gör sidan synlig för användarna.

Id:	**1.6**
Namn:	Redigera en sida
Prioritet:	1
Aktörer:	Administratör
Mål:	Redigera en sida i systemet
Beskrivning:	Administratören väljer att redigera en sida i systemet. Administratören redigerar aktuell information och väljer sedan att spara. Systemet spara redigeringarna.

Id:	**1.7**
Namn:	Ta bort en sida
Prioritet:	1
Aktörer:	Administratör
Mål:	Ta bort en sida från systemet.
Beskrivning:	Administratören väljer att ta bort en hel sida permanent från systemet, sidan går sedan ej att återhämta.

Id:	**1.8**
Namn:	Skapa en fråga
Prioritet:	1
Aktörer:	Administratör
Mål:	Skapa en ny fråga och lägga till den i systemet.
Beskrivning:	Administratören skriver en ny fråga som sedan läggs till i systemet på den plats där administratören vill att den ska vara.

Id:	**1.9**
Namn:	Redigera en fråga
Prioritet:	3
Aktörer:	Administratör
Mål:	Administratören ändrar en fråga eller dess svar.
Beskrivning:	Administratören ändrar text i en fråga, dess svar eller svarsalternativ. Frågan sparas sedan till samma plats i systemet där den fanns innan den ändrades.

Id:	**1.10**
Namn:	Ta bort en fråga
Prioritet:	1

Aktörer: Administratör

Mål: Ta bort en fråga ur systemet

Beskrivning: Administratören väljer en fråga som ska plockas bort från systemet, frågan raderas och kan ej återhämtas.

Id: **1.11**

Namn: Lägg till en övning

Prioritet: 1

Aktörer: Administratör

Mål: Lägga till en övning

Beskrivning: Adminstratören väljer att lägga till en övning. Adminstratörer skapar och lägger till de frågor som ska tillhöra övningen och väljer sedan att spara. Systemet spara övningen och gör den tillgänglig för användarna.

Id: **1.12**

Namn: Ta bort en övning

Prioritet: 1

Aktörer: Administratör

Mål: Ta bort en övning

Beskrivning:	Administratören väljer att ta bort en övning. Administarören väljer att ta bort en övning, frågorna sparas i systemet, men övningen försvinner.

Id:	**1.13**
Namn:	Redigera en övning
Prioritet:	2
Aktörer:	Administratör
Mål:	Redigera en övning
Beskrivning:	Adminstratören väljer att redigera en övning. Adminstratörer byter ordning på frågorna, lägger till nya eller tar bort frågor som tillhör övningen och väljer sedan att spara. Systemet spara redigeringen.

Id:	**1.14**
Namn:	Lägg till en grupp
Prioritet:	3
Aktörer:	Administratör
Mål:	Lägg till en grupp
Beskrivning:	Adminstratören väljer att lägga till en grupp. Adminstratören fyller i gruppinformation och väljer att spara. Systemet spara gruppen och användare går nu att lägga till i gruppen.

Id:	**1.15**
Namn:	Redigera en grupp
Prioritet:	3
Aktörer:	Administratör
Mål:	Redigera en grupp
Beskrivning:	Adminstratören väljer att redigera en grupp. Adminstratöreren redigerar aktuell gruppinformation och väljer sedan att spara. Systemet spara gruppen redigeringarna.

Id:	**1.16**
Namn:	Lägg till en progress check på en sida
Prioritet:	2
Aktörer:	Administratör
Mål:	Lägga till en progress check
Beskrivning:	Adminstratören väljer att lägga till en progress check på en sida. Adminstratören väljer vilka frågor som ska ingå och väljer sedan att spara. System sparar progress checken och gör den tillgänglig för användarna.

Id:	**1.17**
Namn:	Redigera en progress check
Prioritet:	3
Aktörer:	Administratör
Mål:	Redigera en progress check
Beskrivning:	Adminstratören väljer att redigera en progress check. Adminstratören lägger till eller tar bort frågor och väljer sedan att spara. Systemet sparar redigeringarna.

Id:	**1.18**
Namn:	Ta bort en progress check från en sida
Prioritet:	2
Aktörer:	Administratör
Mål:	Ta bort en progress check
Beskrivning:	Adminstratören väljer att ta bort en progress check från vald sida. Systemet tar bort progress checken.

Id:	**1.19**
Namn:	Lägg till en diagnos

Prioritet: 1

Aktörer: Administratör

Mål: Lägg till en diagnos

Beskrivning: Administratören väljer att lägga till en diagnos på en sida.

Adminstratören väljer vilka frågor som ska ingå och väljer sedan att spara. System sparar diagnosen och gör den tillgänglig för användarna.

Id: **1.20**

Namn: Ta bort en diagnos

Prioritet: 1

Aktörer: Administratör

Mål: Ta bort en diagnos

Beskrivning: Adminstratören väljer att ta bort en diagnos från vald sida. Systemet tar bort diagnosen.

Id: **1.21**

Namn: Redigera en diagnos

Prioritet: 2

Aktörer: Administratör

Mål: Redigera en diagnos

Beskrivning: Adminstratören väljer att redigera en diagnos. Adminstratören lägger till eller tar bort frågor och väljer sedan att spara. Systemet sparar redigeringarna.

Id: **1.22**

Ärver: Redigera sida

Namn: Lägg till videoklipp

Prioritet: 3

Aktörer: Administratör

Mål: Lägga till videoklipp

Beskrivning: Adminstratören lägger till ett videoklipp på sidan som redigeras och väljer att spara. Användare kan se videoklippet på sidan.

Id: **1.23**

Namn: Ta bort en grupp

Prioritet: 3

Aktörer: Administratör

Mål: Ta bort en grupp

Beskrivning: Administratören väljer att ta bort en grupp. Det finns användare som tillhör gruppen. Administratören varnas att användare tillhör gruppen. Administratören bekräftar borttagningen och gruppen tas bort. Användarna

som tillhörde gruppen kommer att tillhöra systemets standardgrupp istället.

Id: **1.24**

Namn: Återställ en användare

Prioritet: 3

Aktörer: Administratör

Mål: Återställa en tidigare borttagen användare

Beskrivning: Administratören väljer att återställa en borttagen användare. Systemet återställer användaren och denne kan nu logga in igen.

Id: **1.25**

Namn: Se vilka frågor som tillhör progress-checks

Prioritet: 3

Aktörer: Administratör

Mål: Se vilka frågor som för tillfället kan dyka upp i progress-checks

Beskrivning: Administratören väljer att se vilka frågor som tillhör progess-checks. En sida visas. Om det finns frågor som tillhör progress-checks så visas de i en lista på sidan. I annat fall visas ett meddelande på sidan som säger att det inte finns några frågor som tillhör progress-checks.

Id:	**1.26**
Namn:	Lägga till flera användare
Prioritet:	3
Aktörer:	Administratör
Mål:	Att lägga till fler användare på en gång
Beskrivning:	Administratören väljer att lägga till en grupp användare samtidigt. Administratören laddar in en tabell som sedan läses in och lägger till de användare som finns med i tabellen. Är tabellen tom händer ingenting.

Id:	**1.27**
Namn:	Återställa fråga/diagnos/artikel
Prioritet:	3
Aktörer:	Administratör
Mål:	Att återställa en fråga/diagnos/artikel.
Beskrivning:	Administratören väljer att återställa en fråga/diagnos/artikel. Administratören väljer om den ska hamna den där den tidigare låg eller på en annan plats. Frågan/diagnosen/artikeln återställs om den valda platsen existerar.

Id:	**1.28**

Namn: Strukturera om sidorna i trädstrukturen

Prioritet: 1

Aktörer: Administratör

Mål: Att ändra hur sidorna presenteras genom att ändra deras trädstruktur.

Beskrivning: Administratören väljer att ändra trädstrukturen för sidorna. Administratören låser upp trädstrukturen och drar och släpper sidorna där denne vill ha dem. Administratören väljer att spara ändringarna.

Id: **1.29**

Namn: Visa enskilda användare

Prioritet: 2

Aktörer: Administratör

Mål: Visa en översiktssida för en användare.

Beskrivning: Administratören väljer att visa en profilsida för en användare. Där visas information om epost, grupper och svar på test.

Id: **1.30**

Namn: Visa en sida som en student

Prioritet: 3

Aktörer: Administratör

Mål: Att se en sida som om man vore en student.

Beskrivning: Administratören väljer att visa en sida som en student. Sidan laddas om och visar bara element som en student får se.

Student:

Id: **2.1**

Namn: Genomföra en övning

Prioritet: 1

Aktörer: Student

Mål: Genomföra en kapitel övning för att se hur bra du kan kapitlet.

Beskrivning: Studenten genomför en övning för att testa sina kunskaper i de olika delkapitlen. Studenten genomför frågorna och skickar in. Systemet sparar resultatet och visar det färgkodade resultatet. Systemet visar också kategori för de felaktiga svaren.

Id: **2.2**

Namn: Genomföra progress checks

Prioritet: 2

Aktörer: Student

Mål: Att testa sina kunskaper i progress checks som täcker delar av kursen.

Beskrivning: Studenten testar sina kunskaper i en progress-check som täcker delar av kursen. Studenten genomför frågorna och skickar in. Systemet sparar resultatet och visar det färgkodade resultatet. Systemet visar också kategori för de felaktiga svaren.

Id: **2.3**

Namn: Genomföra diagnos

Prioritet: 1

Aktörer: Student

Mål: Testa sina kunskaper i diagnostiska tester

Beskrivning: Studenten väljer att genomföra en diagnos. Studenten genomför frågorna och skickar in. Systemet sparar resultatet och visar det färgkodade resultatet. Systemet visar också kategori för de felaktiga svaren.

Id: **2.4**

Namn: Se gamla resultat

Prioritet: 3

Aktörer: Student

Mål: Att se sina tidigare resultat

Beskrivning:	En student väljer att se sina resultat från tidigare övningar, diagnoser och progress-checks. Har användaren inte gjort några övningar, diagnoser eller progress-checks så visas ändå en sida fast utan resultat.

Övrigt:

Id:	**3.1**
Namn:	Logga in
Prioritet:	1
Aktörer:	Användare
Mål:	Att autentisiera sig och starta en session.
Beskrivning:	Användaren väljer att logga in och fyller i sina inloggningsuppgifter. Användaren blir autentiserad ifall inloggningsuppgifterna finns i systemet annars nekas användaren tillgång till verktyget.

Id:	**3.2**
Namn:	Logga ut
Prioritet:	1
Aktörer:	Användare
Mål:	Att avsluta en session för en autentiserad användare.
Beskrivning:	Användare väljer att logga ut. Användarkontot loggas ut och sessionen avslutas.

Id:	**3.3**
Namn:	Se sida
Prioritet:	1
Aktörer:	Användare
Mål:	Att se ett kapitel eller en övning.
Beskrivning:	Användaren väljer att visa en sida med ett kapitel eller en övning. Sidan visas sedan för användaren.

Id:	**3.4**
Namn:	Navigera
Prioritet:	1
Aktörer:	Användare
Mål:	Att navigera till en annan sida
Beskrivning:	Användaren väljer att navigera från en sida till en annan. Den nya sidan visas för användaren.

Appendix F - Spårbarhetsmatris

Användningsfall / Krav	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	1.11	1.12	1.13	1.14	1.15	1.16	1.17	1.18	1.19	1.20	1.21	1.22	1.23	1.24	1.25	1.26	1.27	1.28	1.29	1.30	1.31	1.32	1.33	1.34	1.35	1.36	1.37	2.1	2.2	2.3	2.4	3.1	3.2	3.3	3.4
1,1	x																																												
1,2		x																																											
1,3			x																																										
1,4	x																																												
1,5		x																																											
1,6			x																																										
1,7																								x																						
1,8	x	x																																												
1,9				x																																									
1,10																																													
1,11					x																																								
1,12						x																																							
1,13							x																																						
1,14											x																																		
1,15													x																																
1,16												x																																	
1,17																			x																										
1,18																					x																								
1,19																				x																									
1,20								x																																					
1,21									x	x																																			
1,22										x																																			
1,23																											x																		
1,24													x																																
1,25																					x																								
1,26													x																																
1,27																	x																												
1,28																									x																				
1,29																													x															
1,30														x																															
1,31															x																														
1,32																							x																						
1,33																	x																												
1,34																		x																											
1,35																																			x										
1,36																																				x									
1,37																																					x								
2,1																																						x	x	x					
2,2																																													x
2,3																																											x		
2,4																																						x							
2,5																																							x						
2,6																																								x					
2,7																																								x					
2,8																																							x						
2,9																																						x							
2,10																																									x				
2,11																																									x				
2,12																																									x				
3,1																																										x			
3,2																																										x			
3,3																																						x							
3,4	x	x																																											
3,5			x				x																																						
3,6								x	x																																				
3,7								x	x																																				
3,8								x	x																																				
3,9																	x	x																											
3,10																						x																							
3,11					x	x																																							
3,12																																												x		

Appendix G - Entity-relationship diagram över databasen