Mittuniversitetet

MID SWEDEN UNIVERSITY

Metoder och verktyg i mjukvaruprojekt

Lektion 4 – Dokumentation

Lektionen ämnar ge en introduktion till dokumentation av ett projekt!

Innehållsförteckning

1 Doxygen

Det finns många olika verktyg för att generera dokumentation av projektets källkod, för att se en sammanställning rekommenderas artikeln **Comparison of documentation generators** [1]. I praktiken kan det innebära att man i ett stort projekt har olika verktyg för skilda delar. Men med lite eftertanke så som att välja ett verktyg som är plattformsoberoende och som har stöd för många olika programmeringsspråk, så kan man komma ifrån detta. Om sedan verktyget dessutom är *open source* så är det också bra ur många aspekter. **Doxygen** [1] är ett sådant verktyg och har stöd för Windows, Linux och MacOS. Vidare stöder det många *main stream* programmeringsspråk däribland C++, som dessutom använts för att utveckla själva verktyget i sig.

Doxygen extraherar dokumentation från källkodens kommentarer och erbjuder ett sofistikerat syntaxsystem som låter användaren styra innehållet av dokumentationen samt dess formatering via speciella annoteringar. Inställningarna för detta lagras i en konfigurationsfil kallad **doxyfile**, som använder byggsystemet **Make** för att generera den output som bestämts. Du behöver inte sätta dig in i detaljer kring detta byggsystem, men vissa variabler gällande *doxyfile* kan bli aktuellt att hantera manuellt för att möjliggöra samarbete med andra utvecklare.

Vidare stöder *Doxygen* en mängd olika format för den genererade dokumentationen, däribland **HMTL**, **PDF**, **RTF**, **Man Pages** etc. Du kan gärna testa lite olika format men i kursen skall vi huvudsakligen fokusera på *HTML output*.

> *Viktigt att tänka på när det gäller konfigurationsfilen **doxyfile** är att den skall användas av projektets alla deltagare för att generera en aktuell dokumentation. Detta innebär att konfigurationsfilen måste ingå i versionshanteringen, medan dess genererade output skall uteslutas. Eftersom alla deltagare använder olika system så måste också variabeln `OUTPUT_DIRECTORY` anges med relativ sökväg istället för en absolut!*

1.1 Installation

Doxygen kan installeras på två olika sätt, antingen genom att bygga det själv eller genom att ladda ned färdiga binärer [2]. Linuxanvändare kan oftast finna *Doxygen* i distributionens pakethanteringssystem. Verktyget kan sedan användas både som ren CLI eller som via en grafisk front end kallad Doxywizard [3]. Det är en din personliga preferens som avgör vilket gränssnitt som skall användas, men du rekommenderas att installera *Doxywizard* och i alla fall testa den. Båda alternativen erbjuder all nödvändig funktionalitet för att arbeta med *doxyfiles*.

1 http://www.stack.nl/~dimitri/doxygen/
2 http://www.stack.nl/~dimitri/doxygen/download.html
3 https://www.stack.nl/~dimitri/doxygen/manual/doxywizard_usage.html

1.2 Annotering av kod

> *Utförlig dokumentation över syntaxen för att kommentera källkod återfinns i Doxygen's manual [3].*

Vad är det som skall dokumenteras och till vilken grad? Frågor som dessa är inte helt enkla att svara på eftersom det, som så mycket annat, beror på vem som är den tänkta mottagaren av informationen. En *API* behöver inte dokumentera systemets slutna delar utan enbart de öppna gränssnitt som tillåter djupare åtkomst. Medlemmar i ett utvecklingsteam behöver istället en överblick av all *data*, alla *gränssnitt* samt *relationer* mellan entiteter. Normalt brukar sådan information i huvudsak utgöras av diagram men det förekommer även att utvecklingsteam upprättar en intern dokumentation för att understödja samarbetet.

Som tumregel kan man utgå från att alla definitioner av gränssnitt som ger tillträde till systemets slutna delar bör dokumenteras. Detta innebär att *publika klasser* och *funktioner* skall återge sina ansvarsområden för användaren; typ av data som hanteras, interna delar ur programmet som används samt eventuella parameterlistor och returvärden.

Det är viktigt att man följer en konsekvent placering av annoteringarna för dokumentationsgenerering och lämpligen kan allt sådant placeras bland prototyperna i headerfilen, medan *implementationsfilerna* om nödvändigt istället använder vanliga enkla kommentarer. På det sättet sprids dokumentationen inte över olika filtyper och läsaren slipper söka efter denna information. Detta gör det även möjligt att tydligt ge olika **abstraktionsnivåer** beroende på syftet med kommentarerna.

Den information som skall utgöra dokumentationen bör beskriva övergripande koncept och ansvar i termer som inte påverkas av implementationsförändringar. De faktiska detaljerna återspeglas sedan från själva koden, förutsatt att goda **kodkonventioner** efterföljs (se Lektion 1), och som vid behov kan stärkas med vanliga kommentarer. Ett sätt att se detta är att ***implementationen skall vara beroende av definitionen, men aldrig tvärt om***. Den genererade dokumentationen för en metod som hanterar databasåtkomst skall således övergripande beskriva ***vad*** som händer men inte ***hur*** det utförs. Detta minskar beroendet av rådande detaljer och gör det enkelt att exempelvis byta ut en databas mot en annan utan att kräva ändringar i dokumentationen.

> *Nära besläktat till detta är metodologien **Pseudocode Programming Process** (PPP) [2] som används för att designa klasser, moduler samt procedurer. Tanken är att ansvarsområden beskrivs genom en generell pseudokod som inte detaljerar någon implementation och helst ingen språkspecifik syntax, så att även programspråket förblir oberoende. När designen sedan är färdig så kan denna pseudokod användas som dokumentation för entiteten!*

För att illustrera hur *Doxygen* kan användas för att dokumentera programkod använder vi följande exempel som visar en person-klass med innehållande data för namn samt tillhörande konto;

```
/**
 * @file Person.h
 * @author Erik Ström <Erik.Strom@miun.se>
 * @version 1.0
 */

class Person {
  public:
    Person(string name, Account& account) : name(name),
account(account) {}
    Account& getAccount() { return account; }

  private:
    const string name;
    Account& account;
}
```

Som vi kan se innehåller denna kodsnutt inledande kommentarer med annoteringar som relaterar till den aktuella filen. Liknande information bör anges för varje kodfil som tillhör projektet, såväl implementations- som definitionsfiler, och kan även innehålla annan information än det som redovisas här. Andra taggar såsom **`@section`** samt **`@subsection`** kan lämpligen användas för att indela licensdetaljer och annat som kan vara av allmänt intresse inom separata stycken.

Dokumentering av definitioner bör dock undvikas i denna inledning och istället presenteras i direkt anknytning med relevant kod, som vi nu kommer att se lite närmare på.

```
/**
 * @class Person
 * @brief class for holding information regarding a person.
 * @details Each person object represents a member of the organization.
 */
class Person {...
```

Klassdokumentationen består här av de två huvudsakliga taggarna @brief samt @details. Den första återger en kortfattad summering av klassen, och kommer presenteras i relation till entiteten i dokumentationens **`Class List`**. Den sista taggen återfinns i klassreferensen och skall detaljera entitetens ansvarsområde. Båda dessa taggar kan användas till alla typer av entiteter (*class*, *function*, *namespace* etc).

```
/**
 * Constructor that initializes person data.
 *
 * @param name The full name of person.
 * @param account Account details.
 *
 * @code person("Erik Ström", account); @endcode
 */
Person(string name, Account& account) : name(name), account(account) {}
```

Här illustreras hur parametrar kan dokumenteras, men det visar även hur exempelkod kan presenteras genom taggarna **`@code`** samt **`@endcode`**. Nästa kodruta visar hur returvärden skall dokumenteras;

```
/**
 * Method to reference the person's account.
 *
 * @return Reference to account object.
 */
Account& getAccount() { return account; }
```

Det finns många andra taggar som kan vara lämpliga att använda beroende på vilken information som behöver förmedlas. Några som avsevärt kan underlätta samarbetet i ett team är taggarna;

- **`@test`** som återger vad som skall testas i relation till klassen eller funktionen.
- **`@bug`** som redogör för upptäckta felaktigheter i koden eller andra avvikelser.
- **`@todo`** som redogör för vad som behöver läggas till eller åtgärdas.

Förutom att presenteras i klass- och funktionsreferenserna så samlar *Doxygen* alla förekomster från var och en av dessa annoteringar i respektive listor under fliken **`Related Pages`**. Detta innebär att varje deltagare har tillgång till omedelbara överblicksvyer över vad som skall testas, behöver göras samt vilka buggar som hittats.

1.3 Annoteringar för innehållspresentation

Om det föreligger ett behov av mer omfattande beskrivning av koncept eller annan kringliggande information, så kan man lämpligen använda sig av taggen **`@page`** som talar om för *Doxygen* att relaterat textinnehåll skall indexeras som en fristående sida. Vidare kan taggen **`@subpage`** användas för att ytterliggare dela upp relaterat innehåll i egna undersidor. En referens kan sedan anges vart som helst bland övriga annoteringar för att hänvisa läsaren till respektive sida. För att illustrera detta med ett exempel så antar vi att en presentation av projektets medlemmar bör placeras på en egen sida;

```
/**
 * @page about_the_authors This Page presents the project’s authors.
 *
 * @section Author1 Erik Ström
 * @image html daffy.png
 * Information about Erik!
 *
 * @section Author2 Christoffer Fink
 * Information about Christoffer!
 */
```

För att strukturera innehållet på en sida kan man använda taggen **`@section`** som ger en egen rubrik för avsnittet, men om informationen skulle behöva ytterliggare uppdelning kan även **`@subsection`** läggas till. En bild är här inkluderad för den ena deltagaren och för att detta skall fungera måste variabeln **`IMAGE_PATH`** i **Doxyfile** peka på relevant sökväg.

Taggen `@mainpage` används för att fylla information till den huvudsida som genereras av *Doxygen*, och är det som först visas när exempelvis `index.html` öppnas. Denna tag används på ett ställe, lämpligen av `main.cpp` eller annan källa som utgör programmets startpunkt. I stora projekt brukar man dock samla all kringliggande information som skall presenteras som egna sidor i dedikerade header-filer, vilket kan avsevärt underlätta såväl administration som underhåll.

Till **mainpage** lägger man normalt in övergripande information kring projektet samt referenser till övrig information, såsom sidor för medlemspresentationer.

```
/**
 * @mainpage My Project
 * Information about the project!
 *
 * @ref about_the_authors
 */
```

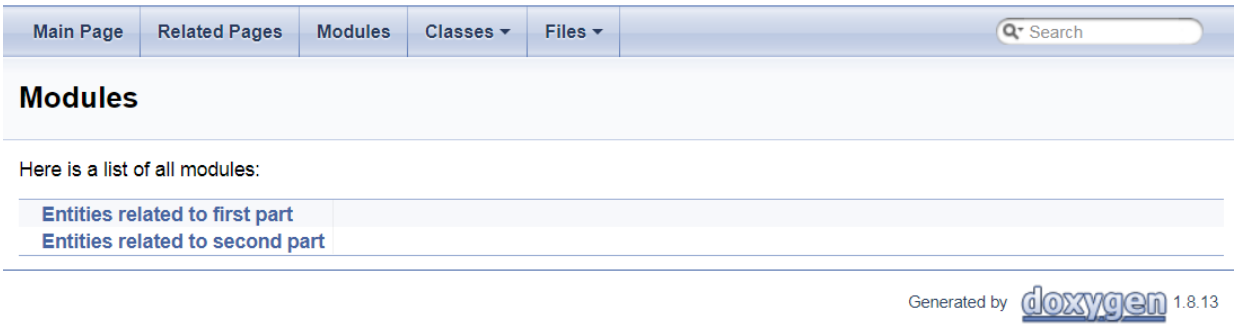
Det finns mycket mer information kring hur innehåll kan formateras och presenteras i Doxygen's manualer. En sak som är extra smidigt är att stylesheets (CSS) används för struktur och layout, och ni kan ange egna styles genom att tala om för variabeln `HTML_EXTRA_STYLESHEET` vart dessa kan hittas. Tänk dock på att användning av såväl egna CSS-filer som bilder ökar det externa beroendet, vilket innebär att dessa måste inkluderas med Doxyfile för att dokumentationen skall kunna återskapas i ett annat system.

För att visa tillhörighet och relationer för och mellan olika entiteter så erbjuder Doxygen möjligheten att placera dessa i sammanhängande grupper / moduler.

```
/**
 * @addtogroup GROUP_A Entities related to first part
 * @{
 */
Class C1 {};
Class C2 {};
/** @} */

/**
 * @addtogroup GROUP_B Entities related to second part
 * @{
 */
Class C3 {};
Class C4 {};
/** @} */
```

Detta kodstycke talar om för Doxygen att entiteterna `C1` och `C2` tillhör `GROUP_A` samt att `C3` och `C4` tillhör `GROUP_B`, och genererar då nödvändig information för dessa under fliken `Modules`;



För att även visa tillhörighet av grupperna i sig så kan man istället nästla dessa som **submoduler** till en huvudgrupp;

```
/**
 * @defgroup MAIN_MODULE
 * @brief Related entities divided into groups!
 *
 * @{
 * @defgroup GROUP_A Entities related to first part
 * @defgroup GROUP_B Entities related to second part
 * @{
 */
```

Sedan lägger man till entiteter för respektive grupp genom att ange dess tillhörighet;

```
/** @ingroup GROUP_A */
class C1 {};

/** @ingroup GROUP_B */
class C3 {};
```

Detta får då följande utseende;

I laboration 3 får du arbeta vidare med generering av dokumentation med hjälp av verktyget Doxygen!

Referenser

[1] Wikipedia, "Comparison of documentation generators", 2017. [Online]
Tillgänglig: https://en.wikipedia.org/wiki/Comparison_of_documentation_generators
[Åtkomst: 17 Juli 2017]

[2] David Zych, "Writing code using the Pseudocode Programming Process", 2013. [Online]
Tillgänglig: https://davidzych.com/writing-code-using-the-pseudocode-programming-process/
[Åtkomst: 12 Oktober 2017]

[3] Doxygen manual, "Documenting the code", 2017. [Online]
Tillgänglig: http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html
[Åtkomst: 20 Augusti 2017]