Schneeweiss Projektorganisationsanweisung für die Softwareentwicklungsarbeiten auf Windows und Mikrocontroller Systemen

Motivation

Softwareorganisationsanweisungen dienen der konsistenten, geordneten und übersichtlichen Aufbereitung von Quellcodes und Projekten. Sie legen einfache, klare Richtlinien zur Aufbereitung, Ablage und Beschreibung von Software fest. Software wird für Programmierer, Projektleitung und Anwender zugänglicher. Programmierern erleichtert sie das Lesen, Verstehen, Warten und Erweitern von Code.

Die Projektleitung gleicht die Projektbeschreibung mit der Dokumentation und den Testresultaten ab. Anwender nutzen exakte Schnittstellen. Coding Standards und ihre sorgfältige Umsetzung haben ein Minimum an Ordnung zur Konsequenz und bilden damit eine wesentliche Grundlage für qualitativ gute Software.

Versionen

Version, Datum	Autor	Änderung
1.0 5.März 2007	Hülsebus	Erste Versionen dieses Dokuments
1.1 25.Juli 2007	Hülsebus	Überarbeitung Projektorganisation

Inhalt

Projektorganisation

Mappen

- Jedem Kunden ist eine Mappe zugeordnet, in der die Projektbezogenen Ordner angelegt sind.
- Jedem Projekt ist eine Mappe zugeordnet, in der sämtliche projektbezogene Dateien gespeichert werden.
- Projektbezogene Dateien sind Pflichtenheft, Besprechungsnotizen, Dokumentation zur Installation und Verwendung der Software und Dokumentation zur Software selbst.
- Weitere Dateien, die hier abgelegt werden, sind Referenzimplementierungen, technische Dokumentationen sowie Software Tools, die ausschließlich für dieses Projekt notwendig sind.

Versionierung

- Projektbeschreibung, Tests, Releases und Dokumentation beziehen sich auf eine bestimmte Version.
- Quellcodes werden regelmäßig, nach jeder relevanten Änderung, im Quellcodeverwaltungssystem eingepflegt. Im Quellcodeverwaltungssystem ist immer die letzte Entwicklungsversion gespeichert. Idealerweise kann aus diesen Kommentaren ein Changelog erzeugt werden, geht das nicht, muss das Changelog per Hand erstellt werden.
- Jede an Kunden ausgelieferte Version ist im Quellcodeverwaltungssystem entsprechend gekennzeichnet.
- Als Quellcodeverwaltungssystem verwenden wir Visual Source Save.

Quellcode

- Der Wurzelordner des Quellcode wird nach dem Projektnamen benannt.
- Pro Namespace (C#) oder Modul (C, C++) ist eine eigene Mappe angelegt. Hier sind Quellcodes und technische Dokumentation zu der Bibliothek zusammengefasst.
- Im Fall von .NET (C#) gibt die die Microsoft Entwicklungsumgebung eine Verzeichnisstruktur für den Quellcode vor. Das Hauptprogramm ist nach dem Projekt benannt.
- Im Fall von C und C++ auf Mikrocontrollern sind die Unterordner `src` für die Quellcode Dateien, `include` für Include Dateien, `doc` für Dokumentation und `lib` für kompilierte Module üblich. Das Makefile und Dokumentationen befinden sich dann im Wurzelordner. Der Name "main" ist für das Hauptprogramm vorbehalten, hier werden ausschließlich Top Level Funktionen und Daten verwendet.
- Ein Quellcodebestand enthält eine Einführung, die den Zweck des Codes kurz beschreibt. Insbesondere enthält das Hauptprogramm eine kurze Einführung, die den Zweck der Software beschreibt und einen Überblick über die Quellcodes verschafft.

Module

- Im Fall von C und C++ auf Mikrocontrollern heißen Header Dateien zu einem Quellcodebestand gleich.
- Gleicht der Quellcodebestandsname dem Modul oder Bibliotheksnamen, so hat dieser Bestand eine zentrale Rolle im Modul oder in der Bibliothek, ähnlich "main" in einem Programm. Ein solcher Bestand enthält, wie "main", eine Einführung, die kurz den Zweck des Moduls beschreibt und einen Überblick über die Quellcodes verschafft.
- Komplexe Datenstrukturen oder Klassen werden in eigenen Modulen definiert. Die Module exportieren den Datentyp in ihrer Schnittstelle.
- Module sind nicht voneinander abhängig. Beispiel: Treiber sind austauschbar um die Software von der Hardware unabhängig zu halten.

Makefiles

- Im Fall von C und C++ auf Mikrocontrollern werden für Makefiles standardisierte Templates angelegt. Soweit möglich werden globale “Variablen“ angelegt, um Pfade und abhängige Bibliotheken im Projekt zu benennen.
- Jedes Modul, jede Bibliothek und jedes Programm erhält ein eigenes Makefile.
- Die Makefiles der Module sind so konfiguriert, dass sie die erstellten Bibliotheken im Ordner „lib“ ablegen.

Daten und Funktionen

- Im Fall von C und C++ auf Mikrocontrollern sind Variablen und Funktionen im Header Bestand und im Quellcode Bestand in der gleichen Reihenfolge sortiert. Im Header Bestand werden nur die exportierten Variablen und Funktionen benannt.
- Im Quellcode Bestand werden erst die "privaten" Variablen und Funktionen definiert, dann die Exportierten. Wenn für "private" Variablen und Funktionen ein Prototyp erforderlich ist, so ist der im Quellcode Bestand angegeben.
- Die Reihenfolge der Statements im Quellcode Bestand ist: include, define, Prototypen, lokale (oder "private") Variablen und Funktionen, globale (oder "public") Variablen und Funktionen.
- Die Trennung dieser Bereiche ist mit einem Kommentar verdeutlicht.
- Variablen und Funktionen sind soweit möglich lokal.

Dokumentation

- Allgemein beschreibt eine Spezifikation der Software in Form eines Pflichtenheftes den Funktionsumfang. Diese Spezifikation bildet die Grundlage für Programmierung und Tests, ist eine Referenz für die Planung und Basis für Erweiterungen.
- Dokumentation wird in Microsoft Word erstellt, diese Dokumente enthalten immer ein Inhaltsverzeichnis. Schaubilder werden in Microsoft Visio erstellt.
- Installation und Verwendung der Software sind für Administratoren und Endanwender dokumentiert.
- Ein spezielles Dokument gibt unter Bezug auf die Projektbeschreibung eine Übersicht über die allgemeine Struktur der Software und der Quellcodes.
- Die Software wird parallel zum Pflichtenheft beschrieben.
- Die Beschreibung von Modulen und Bibliotheken enthält die Spezifikation des Interfaces sowie kurze Beispiele zur Verwendung der einzelnen Funktionen, sollte das aus dem

Kontext nicht klar sein. Das findet im Quellcode jeweils zu den Funktionsprototypen bzw. Klassendeklarationen statt.

- Ein spezielles Dokument beschreibt die exakte Umgebung, Installation und Verwendung für das Projekt, dabei werden Bibliotheken, Compiler, Entwicklungsumgebungen und ggf. auch weitere Software Werkzeuge mit Bezugsquelle, Version und deren Installation benannt. Falls notwendig, werden die Vorgehensweise beim Aufsetzen einer Entwicklungs- und Testumgebung, wie Beispielsweise den Anschluss und die Einstellungen externer Geräte beschrieben.
- Ein spezielles Dokument beschreibt die Änderungen zwischen Versionen anhand der Versionsnummern. Ein gutes Changelog ist eine Undo Liste. Es enthält, per Änderung, deren Datum, Name des Programmierers, die Versionsnummer der Software, sowie kurz und prägnant, die Art der Änderung, die Fehler oder Projektanforderung die die Änderung erforderlich machen und einen Verweis auf den Test der den Erfolg der Änderung dokumentiert. Detaillierte Dokumentation der Änderungen steht nicht im Changelog sondern direkt im Quellcode. Ein Changelog wird nach “Fehler“, “Änderungen, Verbesserungen“, “Neue Funktionalitäten“ usw. sortiert.
- Grafiken wie beispielsweise Flussdiagramme für den Kontrollfluss, Zustandsdiagramme für Automaten, Datenstrukturdiagramme für Strukturen oder Klassen und Schaltpläne für die Belegung und Verwendung von I/O Ports beschreiben gesonderte Zusammenhänge der Software im Detail.
- Dokumentation wird in Deutsch erstellt, ausgenommen davon ist existierende Dokumentation existierender Software, da die üblicherweise in Englisch gehalten wird und ein Umschreiben nicht sinnvoll ist.

Testen

Allgemein

- Tests sollen die Korrektheit der Software sicherstellen. Sie demonstrieren die Anwesenheit, nicht aber die Abwesenheit von Fehlern.
- Testfunktionen werden mit Hilfe der Projektspezifikation erstellt.
- Tests stellen die gestestete Grundfunktionalität zu einer gegebenen Software Version sicher.
- Regelmäßig nach jeder Software Änderung werden alle automatisierten Tests einmal durchgeführt. Dabei sollen die Sourcen aus der Quellcodeverwaltung verwendet werden, um eine eindeutige Zuordnung zwischen Tests und Versionsnummern zu erhalten.
- Tests werden nicht vom Programmierer selbst, sondern von einem Kollegen durchgeführt. Erfolg beim Testen ist ein Misserfolg für den Programmierer.

Labortest

- Der Test im Labor betrachtet das fertige Endgerät, nicht die Software als Solche. Dabei wird unter Zuhilfenahme der Spezifikation überprüft, ob sich das, als Black Box betrachtete, Gerät wie gefordert verhält.
- Labortests werden nicht vom Programmierer, sondern vom Techniker, der das Gerät später verwendet, durchgeführt.
- Ein Spezialfall ist der Labortest mit echten, im Prüfstand gewonnenen, oder simulierten, Daten, die in einzelne Module der Software eingeführt werden. Die Module müssen dafür hardwareunabhängig sein, da die Treiber dann Teil des die Simulation hostenden Systems sind. So lassen sich kostengünstig und sehr effektiv die eigentlich “intelligenten“ Teile der Software testen und weiter entwickeln.

Softwaretest

- Tests werden im Unterverzeichnis “test“ abgelegt.
- Testfunktionen werden wie folgt benannt: Test_[Funktionname]_[Spezialisierung des Tests]

```
Test_I2cLeseDaten_Gueltig ()
Test_I2cLeseDaten_Ungueltig ()
Test_I2cLeseDaten_Bereich ()
```

- Die Qualität des Codes wird mit einem geeigneten Tool wie QA-C oder lint überprüft. Damit wird auch eine weitgehende Einhaltung des Coding Standards getestet.
- Mit Hilfe von #define Direktiven (in C und C++) bzw. dem Debug Namespace (C#) werden Debug Level erstellt um einfach zwischen Test und Produktionsversionen umschalten zu können.
- Software wird systematisch und weitgehend automatisiert getestet. Testfälle für Unit Tests werden mit einem geeigneten Tool wie Tessy automatisch erstellt und gemeinsam mit den Quellcodes in der Quellcodeverwaltung versioniert.
- Jedes Modul, Klasse und Funktion wird jeweils als “Blackbox“ und “Whitebox“ betrachtet. Das gilt nicht für Funktionen aus Fremdbibliotheken, die als getestet angesehen werden

müssen. Qualitätsmängel in verwendeten Bibliotheken werden beschrieben und softwaretechnisch umschifft.

- Bei Blackbox Tests wird nach Ein und Ausgangsparametern getestet. Dabei wird die an die Projektbeschreibung angelehnte Spezifikation der Ein und Ausgabemengen herangezogen. Die auch im Funktionskopf kommentiert ist. Getestet werden Elemente aus der Menge sowie Grenzwerte innerhalb und außerhalb der Menge. Geprüft wird die Korrektheit der Berechnung und das Eintreten erwünschter Seiteneffekte.
- Whitebox Tests sind im Grunde Blackbox Tests im Kontext des Funktionsaufrufes (wie ein Breakpoint). An definierten Stellen im Code werden die Zustände der Software geprüft. Dabei werden die in der Projektbeschreibung festgelegten korrekten Werte und Zustandsübergänge herangezogen.

Vorgehensweise

- Grenzwert Tests testen Eingabewerte an den Grenzen der gültigen Eingabemenge. Beispiel: für Fak () werden die Werte -1, 0, 1, 15, 16, 17 und je ein beliebiger Wert aus den drei möglichen Eingabemengen getestet: n für “n < -1“, für “0 <= n <= 16“ und für “n > 17“.
- Vor und Nachbedingungen kennzeichnen einen Zustand der Software. Beispiel:

```
int Fak (int n)
{
    if (n == 0)
    {
        // 0! = 1, also Abbruch der Rekursion
        return (1);
    }
    else
    {
        // n! = n * (n - 1)! für n > 0
        return (n * Fak (n - 1));
    }
}
```

Vorbedingung ist hier “0 <= n <= 16“, Nachbedingung ist: “Fak (n) = n!“. Das Verhalten der Funktion für n < 0 ist im Sinne des Testens nicht definiert, da die hier entstehende Endlosrekursion in der Projektbeschreibung keine Vorgabe darstellt.

- Asserts stellen Vorgaben zur Laufzeit sicher, Beispiel:

```
int Fak (int n)
{
  assert (n >= 0);
  ...
```

- Situationen, die „nicht passieren sollten“ werden abgefangen, Beispiel:

```
int Fak (int n)
{
    if (n <= 0)
    {
        // 0! = 1, also Abbruch der Rekursion
        return (1);
```

```
}
...
```

- Tests sollen jeden möglichen Code Pfad für jede mögliche Eingabemenge abdecken. Das wird durch geschickte Mischung aus Blackbox und Whitebox Tests erreicht. Begonnen wird bei der kleinstmöglichen Blackbox, in der Regel Bibliotheksfunktionen. Ist eine selbstgeschriebene Funktion, die diese Blackbox verwendet und jetzt als Whitebox behandelt und gestestet wird, vollständig erfasst und korrekt, so wird sie zu einer neuen Blackbox.
- Bei der Code Pfad Analyse wird für jeden Code Pfad ein Testfall erzeugt, der prüft, ob alle Werte in den Vorgesehenen Wertemengen liegen. Dabei wird der erste Testfall am Funktionseingang eingesetzt. Für jedes Statement, dass ein “if“, ein “while“, ein “for“, ein “repeat“, ein “and“, oder ein “or“ enthält, wird ein weiterer Testfall erzeugt. Für jedes “switch“ in einem “case“ Statement wird ebenfalls ein Testfall erzeugt. Die Menge der Eingaben in die Funktion ist dann so zu wählen, dass jeder Testfall in der Funktion einmal erreicht wird.
- Bei der Daten Pfad Analyse wird für jede Variable der Moment der Definition (also Initialisierung oder Zuweisung) und jede Verwendung (als Argument in Ausdrücken) mit einem Testfall auf Gültigkeit überprüft.