

```

        extract_number_and_incr(destination, source) int
        *destination; unsigned char **source; { extract_number_and_incr(destination, *source); *source += 2; } #ifndef EXTRACT_MACRO
        ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUMBER_AND_INCR(dest, src) extract_number_and_incr (&dest, &src) #endif /*
        not EXTRACT_MACROS */ #endif /* DEBUG */ /* If DEBUG is defined, Regex prints
        many voluminous messages about what it is doing (if the variable 'debug' is nonzero). If
        linked with the main program in 'iregex.c', you can enter patterns and strings interactively.
        And if linked with the main program in 'main.c' and the other test files, you can run the al-
        ready-written tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h>
        /* It is useful to test things that "must" be true when debugging. */ #include <assert.h> static int
        debug = 0; #define DEBUG_STATEMENT(e) #define DEBUG_PRINT1(x) if (debug) printf (x) #define
        DEBUG_PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
        (x1, x2, x3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
        BUG_PRINT_COMPILED_PATTERN(p, s, e) if (debug) print_partial_compiled_pattern (s, e) #define DE-
        BUG_PRINT_DOUBLE_STRING(w, s1, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, sz1, s2, sz2)
        extern void printchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap)
        char *fastmap; { unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++]
        { was_a_range = 0; printchar (i - 1); while (i < (1 << BYTEWIDTH) && fastmap[i]) { was_a_range = 1; i++; } if
        (was_a_range) { printf ("-"); printchar (i - 1); } } putchar ('\n'); /* Print a compiled pattern string in hu-
        man-readable form, starting at the START pointer into it and ending just before the pointer END. */ void
        print_partial_compiled_pattern (start, end) unsigned char *start; unsigned char *end; { int mcnt, mcnt2; un-
        signed char *p = start; unsigned char *pend = end; if (start == NULL) { printf ("(null)\n"); return; } /* Loop over
        pattern commands. */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
        break; case exactn: mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); printchar (*p++); }
        while (--mcnt); break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
        *p++); break; case stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
        break; case duplicate: printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
        break; case charset: case charset_not: { register int c; printf ("/charset%s", (re_opcode_t) *(p -
        1) == charset_not ? "_not" : ""); assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
        unsigned char map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
        (map_byte & (1 << bit)) printchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
        line: printf ("/begline"); break; case endline: printf ("/endline"); break; case on_failure_-
        jump: extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
        break; case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
        ("/on_failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
        tract_number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
        case push_dummy_failure: printf ("/push_dummy_failure"); break; case may-
        be_pop_jump: extract_number_and_incr (&mcnt, &p); printf
        ("/maybe_pop_jump/0/%d", mcnt); break; case pop_failure_-
        jump: extract_number_and_incr (&mcnt, &p); printf ("/pop_-
        failure_jump/0/%d", mcnt); break; case jump_past_alt:
        extract_number_and_incr (&mcnt, &p); printf ("/-

```

JAN HEIMER

WIE DIGITALE PRODUKTE FÜR KUNDEN
INDIVIDUALISIERBAR BLEIBEN

ENTWICKLUNG EINES CUSTOMIZATION
FRAMEWORKS FÜR CLOUDBASIERTE
SHOPFLOOR MANAGEMENT SYSTEME

Jan Heimer

**Entwicklung eines Customization
Frameworks für cloudbasierte
Shopfloor Management Systeme**

**Wie digitale Produkte für Kunden
individualisierbar bleiben**

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Impressum:

Copyright © Studylab 2020

Ein Imprint der GRIN Publishing GmbH, München

Druck und Bindung: Books on Demand GmbH, Norderstedt, Germany

Coverbild: GRIN Publishing GmbH | Freepik.com | Flaticon.com | ei8htz

Abstract

Digitale Produkte stellen sich im Kontext der Industrie 4.0 der Herausforderung, dass zur erfolgreichen Integration des Systems in ein Unternehmen meist ein hoher Grad an kundenspezifischer Individualisierung in Form von funktionalen und technischen Anpassungen notwendig ist. Dadurch wird für den Kunden die angestrebte Digitalisierung langwierig und kostenintensiv. Für den Anbieter herrscht Planungsunsicherheit und es ist schwierig Skalenerträge mit dem Produkt zu erzielen.

Ziel dieser Arbeit ist es ein Customization Framework für ein digitales Shopfloor Management System am Beispiel des Startups SFM Systems GmbH herzuleiten, um dem Grad an Skalierbarkeit, Standardisierung und Flexibilität auch bei einer cloudbasierten Systemarchitektur gerecht zu werden.

Hierbei werden auf Basis des menschenzentrierten Gestaltungsprozesses nach DIN EN ISO 9241-210 eine funktionale und architektonische Modularisierung und eine standardisierte Anforderungsadaption eingeführt, die es dem Kunden ermöglichen, mittels eines Konfigurators das System flexibel seinen Wünschen und Bedürfnissen anzupassen.

Dadurch wird mit Hilfe des Customization Frameworks die methodische Grundlage gebildet, eine klassische On Premise Shop Floor Management Lösung in einen standardisierten, flexiblen und skalierbaren Cloud Service zu überführen.

Inhaltsverzeichnis

Abstract	III
Inhaltsverzeichnis	IV
Abbildungsverzeichnis	VI
Tabellenverzeichnis	VII
Abkürzungsverzeichnis	VIII
1 Einleitung und Motivation.....	1
1.1 Ziel der Arbeit	1
1.2 Methodik und Aufbau der Arbeit.....	2
2 Grundlagen zur Thematik.....	4
2.1 Definition Digitalisierung.....	4
2.2 Definition Industrie 4.0.....	5
2.3 Cloud Computing	8
3 Herleitung der Framework-Methodik.....	12
3.1 Grundsätze der menschenzentrierten Gestaltung	12
3.2 Phasen und Gestaltungsaktivitäten	14
3.3 Abgeleitete Framework-Struktur.....	19
4 Nutzenkontext	22
4.1 Systemumgebung	22
4.2 Zielgruppe.....	24
4.3 SFMS Digital Teamboard	27

5 Anforderungsermittlung.....	32
5.1 Modularisierung.....	32
5.2 Standardisierung der Anforderungsadaption	43
6 Implementierung.....	49
6.1 Eigenschaften des Konfigurators	49
6.2 Konfigurationsformular	50
6.3 Auswertung und Ergebnis des Konfigurators.....	51
7 Evaluation.....	53
7.1 Ergebnis für die Groß AG	53
7.2 Ergebnis für die KMU GmbH	56
7.3 Ergebnis für die Klein GmbH.....	57
7.4 Kritische Würdigung der Ergebnisse.....	58
8 Fazit und Ausblick.....	60
8.1 Ausblick.....	61
Anhang	62
Literaturverzeichnis.....	65

Abbildungsverzeichnis

Abbildung 1: Aufbau der Arbeit in Anlehnung an den menschenzentrierten Gestaltungsansatz.....	2
Abbildung 2: Auflösung der Hierarchieebenen im Produktionssystem.....	7
Abbildung 3: Kerntechnologien von Industrie 4.0	8
Abbildung 4: Cloudarchitektur mit Betreibermodellen.....	10
Abbildung 5: Phasen der menschenzentrierten Gestaltungsaktivitäten aus DIN EN ISO 9241-210	15
Abbildung 6: Zusammenhang zwischen DIN 9241-210 und dem SFM Customization Framework	21
Abbildung 7: Framework-Module der Nutzenkontext Phase	22
Abbildung 8: Übersicht der allgemeinen DTB Systemarchitektur für eine Teaminstanz.	29
Abbildung 9: Framework-Module der Anforderungsermittlung	32
Abbildung 10: Modell der Produktstruktur mit verschiedenen Individualisierungsstufen	33
Abbildung 11: Architekturvarianten	39
Abbildung 12: Framework-Module der Implementierungsphase.....	49
Abbildung 13: Funktionsmechanik des Konfigurators.....	51

Tabellenverzeichnis

Tabelle 1: Ableitungen aus der DIN ISO 9241 für das Customization Framework.....	20
Tabelle 2: Übersicht Persona.....	26
Tabelle 3: Binäre Design Structure Matrix der DTB Module	37
Tabelle 4: Qualitative Eigenschaftsmatrix zum Vergleich der Architekturvarianten	43
Tabelle 5: Tableau zur Funktionsumfangsadaption auf Basis der erweiterten Design Structure Matrix.....	45
Tabelle 6: Mit p_i gewichtete Eigenschaftsmatrix der Architekturvarianten.....	48
Tabelle 7: Elemente des Konfigurationsformulars zur Erhebung relevanter Informationen.....	50

Abkürzungsverzeichnis

API	Application Programming Interface (dt. Programmierschnittstelle)
CiP	Zentrum für industrielle Produktivität
CPS	Cyber-Physisches System
CSV	Comma-separated values (Dateiformat)
DSM	Design Structure Matrix
DTB	Digitales Teamboard
ERP	Enterprise Ressource Planning (dt. Geschäftsressourcenplanung)
IaaS	Infrastructure as a Service
IKT	Informations- und Kommunikationstechnologie
in-GmbH	in-integrierte Informationssysteme GmbH
IoT	Internet of Things
IS	Information Systems
KPI	Key Performance Indicator (dt. Leistungskennzahl)
MES	Manufacturing Execution System (dt. Produktionsleitsystem)
MIT	Massachusetts Institute of Technology
NIST	National Institute of Standardization and Technology
OPC UA	Open Platform Communications Unified Architecture
PaaS	Platform as a Service
PoC	Proof of Concept
PTW	Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen
REST	Representational State Transfer
SaaS	Software as a Service