Big Data im Nanobereich: Halbleiter ICs mit Oracle, R & Co.

Peter Czerner
Elmos Semiconductor AG
Dortmund

Schlüsselworte

Halbleiter, Oracle, R, Shiny

Einleitung

Ein Halbleiter-Chip durchlebt bei seiner Entstehung etwa 500 Produktionsschritte. Die Komplexität jedes einzelnen ist oft mit der eines mittelgroßen Unternehmens vergleichbar, insbesondere in Bezug auf Planung, Logistik, Datenaufkommen, Steuerung etc.

Wir bei Elmos Semiconductor AG planen, produzieren und steuern mit Daten. Der Vortrag stellt einige unserer Lösungen vor: Datenaufbereitung, Visualisierung, Machine Learning und Ansätze zu Big Data realisiert auf einer auf Oracle und R basierenden Kernarchitektur. Wir schließen die Brücke zwischen Oracle-Datenbank und R. Wir erklären, was eine Wafer-Map ist und wie ihre Darstellung mit wenigen Zeilen R- und SQL-Codes realisiert werden kann. Wir machen einen Ausflug in einen Wald voller Entscheidungsbäume und zeigen, wie wir dort Ursachen für Ausbeuteverluste finden. Wir suchen mit einer Shiny-Webanwendung in über 10.000 Anlagenparametern nach Auffälligkeiten, per Knopfdruck und in wenigen Minuten...

Bei unseren Datenanalysewerkzeugen verbinden wir absichtlich die bewährte Oracle-Datenbanktechnologie mit neuen, meist frei verfügbaren Methoden. Wir versuchen anschließend zu erläutern, wieso es für ein produzierendes Halbleiterunternehmen oft effektiver ist, eigene Software zu bauen.

Vom Wafer zum Chip: Halbleiterfabrik und jede Menge Daten

Die Elmos Semiconductor AG stellt kunden- und applikationsspezifische Halbleiterchips und Sensoren her. Die Komplexität eines Halbleiterchips, sein Design, Produktionsprozesse, hunderte Produktionsschritte und eine Null-Fehler-Qualität sind unser tägliches Geschäft. Gleichermaßen herausfordernd und aber zwingend notwendig ist die Herstellung und Betriebssicherung der Datenhaltung und Datenauswertung hinter den Herstellungsprozessen. Diese werden in Datenmodellen abgebildet, durch Softwareprozesse gelenkt, durch Datenanalyse überwacht und optimiert.

Ein relationales Datenbankmodell auf Basis von Oracle Software gewährleistet bei Elmos schon seit über 20 Jahren eine sichere Datenhaltung. Es stellt über die reine Datenspeicherung weit hinausgehende Prozessabbildungen konsistent und betriebssicher zur Verfügung [1].

Um eine Vorstellung über die Datenmenge zu erhalten betrachten wir folgendes: Elmos produziert etwa 300 Mio. Halbleiter pro Jahr [6]. Jeder davon liefert bis zu 9000 unterschiedliche Messwerte allein beim Test. Dazu kommen Anlagen-, Prozess- und Bewegungsdaten von bis zu 500 Arbeits- und Testschritten. Alles zusammen macht vielleicht keinen Weltrekord aus, aber wenn man die Vielfalt der

Daten und die Komplexität derer Zusammenhänge betrachtet wird einem schnell klar, dass man bei der Auswertung mit SQL, Excel und ähnlichem Werkzeug alleine nicht weit kommt.

Vor einigen Jahren haben wir uns deshalb für R entschieden. R ist eine freie Programmiersprache für statistische Auswertungen und deren grafische Darstellung [2]. Die überwiegenden Kriterien bei der Entscheidung waren die Verbreitung, die Internetpräsenz und vor allem die Unmenge an zur Verfügung stehenden statistischen Verfahren die mit R frei Haus geliefert werden.

R ließ sich auf Anhieb in unsere IT Infrastruktur integrieren. Dabei nutzen wir das ROracle Package für die Ankopplung an unsere Oracle Datenbanken und Shiny [8] für die Erstellung der Webanwendungen. Für die Entwicklung benutzen wir als IDE das in dem Umfeld gleichermaßen populäre wie auch nützliche RStudio [3].

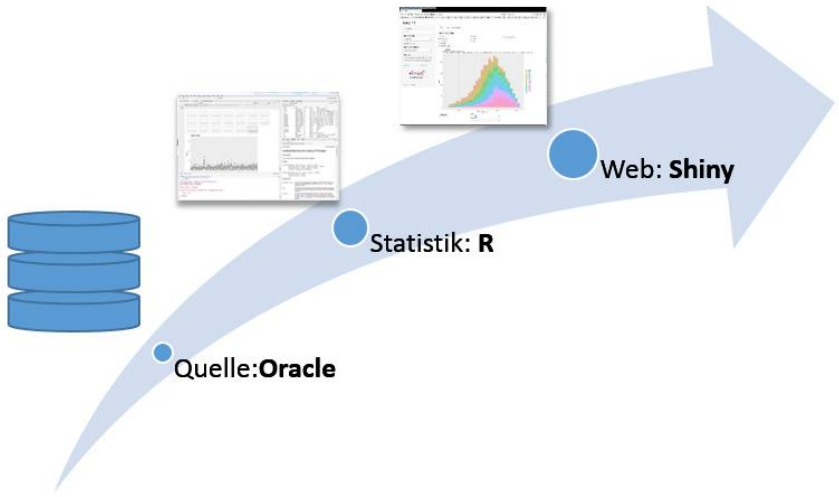

Abb. 1: Toolchain für statistische Datenauswertung bei Elmos

Lassen Sie mich an der Stelle etwas konkreter werden…

R passt perfekt in die IT Landschaft oder: Wafermap in 4 Zeilen

Halbleiterchips werden auf Wafer produziert. Das sind Scheiben aus monokristallinen Silizium, die etwa ein Millimeter dick sind. Eine Wafermap ist ein digitales Abbild eines Wafers. Sie enthält alle zur Verfügung stehenden Informationen über einen Produktions-Wafer. Beispielweise ist die Position jedes Chips auf dem Wafer gespeichert, seine Abmessungen, seine eindeutige Identifikationsnummer und die Information, ob dieser Chip nach allen durchgeführten Tests innerhalb der vorgegebenen Parameter funktioniert. Andersherum wird die Identifikationsnummer im RAM jedes Chips gespeichert, so dass man - sogar nachdem ein Wafer zersägt und die Chips in ihrem Gehäuse und später in der finalen Anwendung verbaut sind – immer noch feststellen kann, auf welchen Wafer und an welcher Position ein konkreter Chip sich befunden hat. Diese Eigenschaft auch „Die Traceability“ (Chip Verfolgbarkeit) genannt ist grundlegend für Null-Fehler-Qualität.

Die Wafermap enthält also zahlreiche nützliche Informationen, die den Fachleuten in der Produktion zur Verfügung gestellt werden müssen. Eine grafische Darstellung ist dafür die beste Möglichkeit. Diese werden wir im Folgenden Schritt für Schritt erläutern.

Zuerst müssen wir die Daten von der Oracle Datenbank nach R transferieren. Hier wirken mehrere R Packages zusammen:

1. **ROracle –** OCI Interface von Oracle [8]
2. **DBI –** Standard Datenbank Interface [10]
3. **Oracle** – Selbstentwickelte Erweiterung, die die Definition der Datenbankverbindungen und eine etwas komfortablere Schnittstelle für Bind-Variablen enthält. Letzteres ermöglicht uns die SQL Abfragen mit PL/SQL Developer [12] zu entwickeln und zu testen, indem sie die gleiche Syntax für die Bind-Variablen benutzt. Das Package führt ebenfalls gespeicherte Abfragen aus, so dass diese von mehreren Auswertungen oder Anwendungen benutzt werden können.

Zusätzlich benötigen wir noch
4. **ggplot2 –** Eine sehr mächtige Grafikbibliothek [11]

Mit diesen Hilfsmitteln – 1,2 und 4 übrigens quelloffen und frei im Internet verfügbar – programmieren wir die Visualisierung einer Wafermap in vier Zeilen:

```
library(oracle)
library(ggplot2)
mapData<-sql("select * from wafermap where lot=123456 and wafer_nr=13")
ggplot(mapData, aes(x=X, y=Y)) + geom_polygon(aes(fill=TYPE, group=DIENUM), size=0.1, color="black")
```

Skript 1: Wafermap in 4 Zeilen

Die ersten zwei Zeilen laden die nötigen Packages (das „oracle“ Package lädt ROracle und DBI). In der dritten Zeile werden die Daten aus der Datenbank per SQL Abfrage geladen.

LOT <dbl>	WAFER_NR <dbl>	ORDERNR <dbl>	DIENUM <dbl>	X <dbl>	Y <dbl>	TYPE <chr>
123456	13	231	229	-17220	-91390	edge
123456	13	232	229	-17220	-89100	edge
123456	13	233	233	-17220	-89100	die
123456	13	234	233	-17220	-91390	die
123456	13	235	233	-14350	-91390	die
123456	13	236	233	-14350	-89100	die
123456	13	237	237	-14350	-89100	die
123456	13	238	237	-14350	-91390	die
123456	13	239	237	-11480	-91390	die
123456	13	240	237	-11480	-89100	die

231-240 of 18,640 rows Previous 1 … 22 23 24 25 26 … 100 Next

Tab. 1: Datentabelle (data frame) in R, wie sie mit einer SQL Abfrage für eine Wafermap extrahiert wurde.

Zu guter Letzt wird in der vierten Zeile die Wafermap mit ggplot gezeichnet, dafür liefern wir mit der SQL Abfrage für jeden Chip 4 Eck-Koordinaten (siehe Tab. 1) und stellen sie als Polygon dar. In Abb. 2 ist das Resultat unseres Skriptes zu sehen.
Jeder Chip der Wafermap ist als Viereck dargestellt, dabei markieren die unterschiedlichen Farben unterschiedliche Chip-Typen: „die“ ist ein normaler Chip, „edge“ ist ein Rand-Chip, der sich zu nah am Rand befindet und als Qualitätssichicherungsmaßnahme aussortiert wird. Mit dieser Methode öffnen sich unzählige Möglichkeiten: Erweitern wir die SQL Abfrage so, dass eine zusätzliche Spalte mit einem Messwert (einen pro Chip) hinzugefügt wird, können wir die Messwertverteilung über den Wafer als „Wärmemap“ darstellen (siehe Abb. 3).

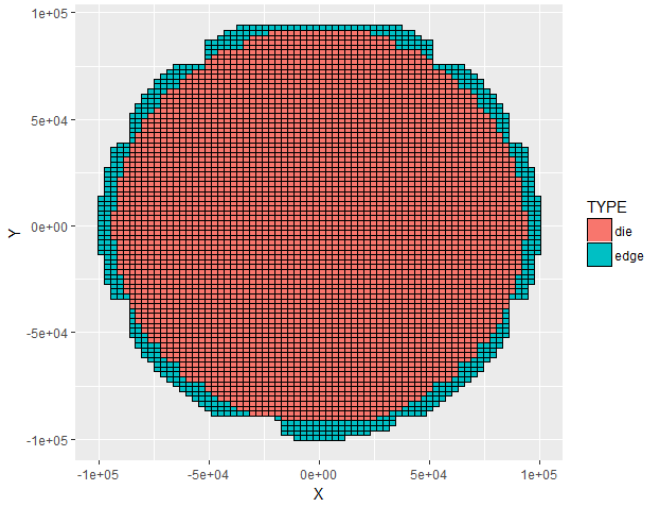

Abb. 2: Wafermap erstellt in 4 Zeilen Code.

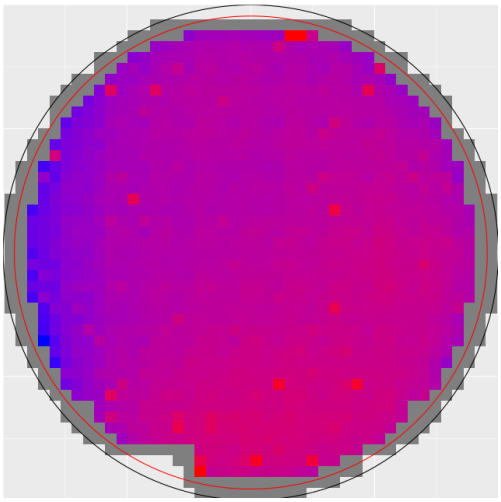

Abb. 3: Farbliche Darstellung eines Messwerts

Der Informationsgehalt einer solchen Darstellung ist gewaltig: Auf einen Blick erkennt man, ob der Messwert homogen über den Wafer verteilt ist und welche Chips aus der Reihe tanzen. Damit ist eine Diagnose in Problemfällen viel einfacher und effizienter.

Shiny macht es möglich: Effizient Data Mining Anwendungen bauen

Wenn ich Sie bis jetzt von der Einfachheit und Effizienz unseres Ansatzes nicht überzeugen könnte, werde ich in diesem Kapitel sicher das entscheidende Argument liefern. Dazu erweitern wir das Skript aus dem letzten Kapitel zu einer interaktiven Webanwendung. Hierfür laden wir das Package Shiny und erstellen per R-Skript eine Web-GUI (siehe Skript 2).

```
1  library(shiny)
2  library(oracle)
3  library(ggplot2)
4
5  ui <- fluidPage(
6     # App title
7     titlePanel("Wafermap for DOAG"),
8
9     sidebarLayout(
10       # Define entries
11       sidebarPanel(
12          numericInput("lot", "Lot: ",min = 1, max = 9999999,value = 123456),
13          numericInput("wafer", "Wafer: ",min = 1, max = 25,value = 13),
14          hr(),
15          HTML('<center><img src="DataScience.png"></center>')
16       ),
17
18       # Define plot frame
19       mainPanel(
20          plotOutput("wafermap",width = "480px", height = "400px")
21       )
22    )
23 )
24
```

Skript 2: GUI Teil der Wafermap-Web-Anwendung

Hier werden die zwei Eingabe-Tags definiert (Zeilen 12 und 13) wie auch das Diagrammfenster, das die Wafermap beherbergt. Damit wäre die GUI fertig, siehe Abb. 4

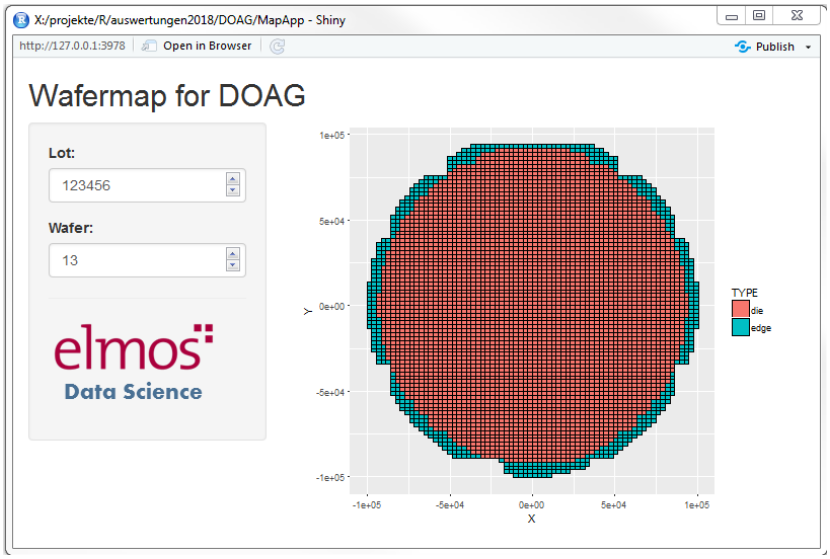

Abb. 4 Wafermap Web-Annwendung

Die Server-Logik hinter der Anwendung (siehe Skript 3) enthält im Grunde genommen nicht viel mehr als unser erstes Skript, außer dass die SQL Abfrage parametrisiert ist (Zeile 27 und 28) und dynamisch die Eingabewerte der GUI annimmt. Die Methode `renderPlot` (Zeile 26) ist ein sog. Reaktiver-Element, d.h. die Event-Behandlung hinter der GUI wird in diesem Fall automatisch geliefert: Sobald eine der Eingaben geändert wird, werden die Daten neu geladen und das Bild neu aufgebaut. Das Gute an Shiny ist, dass die Webanwendungen per R-Skript programmiert werden. Damit erübrigt sich – natürlich wie immer in solchen Fällen nur bis zur einem gewissen Grade – die Kenntnis von HTML und JavaSkript.

```
24
25  # Define server logic required to draw a wafermap
26  server <- function(input, output) {
27
28     output$wafermap <- renderPlot({
29       mapData<-sql("select * from wafermap where lot= &lot and wafer_nr= &wafer",
30                    bindData = data.frame(lot=input$lot, wafer=input$wafer))
31       ggplot(mapData, aes(x=X, y=Y)) +
32         geom_polygon(aes(fill=TYPE, group=DIENUM), size=0.1, color="black")
33     })
34  }
35
36  # Run the application
37  shinyApp(ui = ui, server = server)
38
39
```

Skript 3: Server-Logik der Wafermap-Webanwendung

Damit haben wir unsere Wafermap als Webanwendung in 37 Zeilen realisiert. Diese kann man direkt in RStudio starten und testen. Wenn wir die Anwendung für andere Benutzer zur Verfügung stellen möchten - was meistens der Sinn der Übung ist - braucht man allerdings einen Shiny Server. Hier nur kurz erwähnt: Dessen Installation war in unserem Falle ziemlich unkompliziert und auch der Betrieb mit mehreren Benutzern und Anwendungen verlief bis jetzt unproblematisch.

Geht noch mehr? Klar: Interaktive Wafermap

Mittlerweile haben wir bei Elmos mehrere Wafermap-Anwendungen: Zur grafischen Darstellung der Testwerte und Partikel Messergebnisse, zur Darstellung der Ausfälle für mehrere Wafer einer Charge. Aktuell arbeiten wir an einer Anwendung, die anhand einer Wafermap Anlagen- und Prozessprobleme darstellt und Hinweise auf Ursachen gibt. Dazu ist eine interaktive Wafermap nötig, in der man einzelne Chips oder Chip-Gruppen markieren kann und anhand dieser Auswahl weitere Auswertungen erstellt, z.B. Statistiken, Trendgrafiken oder auf Wahrscheinlichkeitsverteilung und Feature Selection basierende Ursachenanalyse.
Die Wafermap implementieren wir mit dem Package Leaflet, das eigentlich für Kartenanwendungen gedacht war, sich allerdings für unsere Zwecke wunderbar eignet.

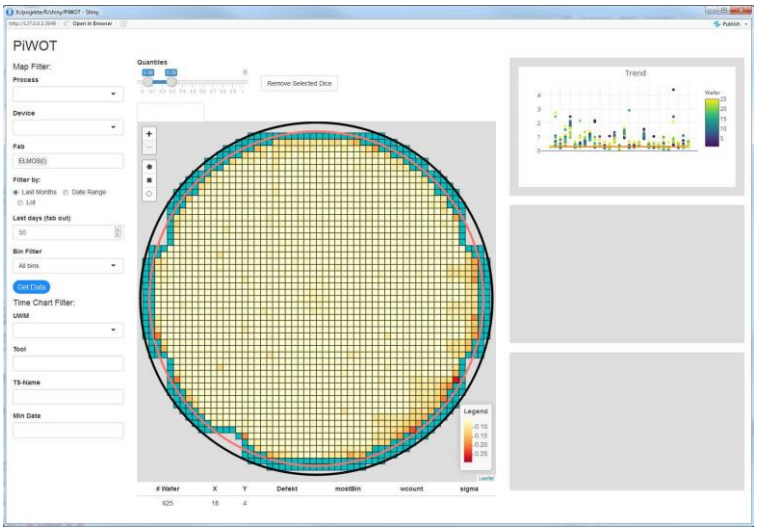

Abb. 5: Interaktive Wafermap-App mit Shiny und Leaflet

Ein Teufel im Wald: Feature Selection für die Ursachenanalyse

Betrachten wir das folgende Szenario: Ein Messwert, z.B. vom Funktionaltest driftet aus dem gewöhnlichen Wertebereich oder es gibt ungewöhnliche Ausbeuteverluste für einige Wafer eines Produktes. Die Fragen, die in solchen Fall unsere Ingenieure beschäftigen sind die Folgenden: Was ist in diesem Falle anders? Welche Anlage ist ursächlich für den Drift verantwortlich?

Das Problem bei der Suche nach Antworten besteht darin, dass es bei 500 Prozessen und etwa 200 Anlagen tausende mögliche Wegkombinationen gibt, die durchsucht werden müssen. Ein Data Scientist spricht hier von einem Ereignisraum mit 200 Dimensionen. Manuell eine sehr aufwändige Angelegenheit, die in 80% der Fälle von erfahrenen Fachleuten mit Prozesswissen deutlich reduziert und schnell gelöst werden kann. Es bleiben jedoch etwa 20% schwierigere Fälle, die mehrere Fachleute über Tage beschäftigen.

In diesen Fällen hat uns oft eine Methode namens Feature Selection [9] geholfen. Es handelt sich dabei um eine Methode aus dem Bereich des maschinellen Lernens, die zur Reduktion der Dimensionszahl verwendet wird. Die Vorgehensweise hier ist wie folgt:

1. **Datenbeschaffung**: Daten mit betroffenen und nicht betroffenen Wafer mit allen Prozessen als

Dimensionen werden zusammengestellt. Als Wert nehmen wir problemabhängig den Anlagennahmen, den Rezeptnamen oder auch den Zeitstempel des Prozesses. Dazu kommt ein Ergebnisvektor, d.h. der Wert des betroffenen Funktionaltests.
2. **Datenaufbereitung**: Alle Daten werden in ein Data Frame gepackt. Die nicht vorhandenen Werte müssen aufgefüllt oder entfernt werden, die nicht numerischen Werte durch geeignete Abbildung wie z.B. Faktorisierung in Zahlen überführt werden.
3. **Ausführung** der Feature Selection Methode. Als Resultat erhält man eine reduzierte Liste der Prozesse, die als Ursache für das Problem infrage kommen, zusammen mit einem Ranking.
4. **Verifikation**: Die so erhaltenen Ergebnisse müssen verifiziert, d.h. mit geeigneten Methoden verifiziert oder wiederlegt werden.

Als konkrete Implementierung haben wir das Boruta Package [3] verwendet. Es handelt sich dabei um einen Algorithmus, der auf Random Forest basiert (Der Name „Boruta“ stammt von einem Waldteufel aus einer polnischen Sage). Um die Vorgänge zu vereinfachen, haben wir die Schritte in einem Package implementiert, so dass eine Auswertung mit wenig Aufwand erfolgen kann. Dabei ist die Datenbeschaffung besonders wichtig, da sie immer die meiste Zeit der Auswertung in Anspruch nimmt. Wir haben versucht, verschiedene Datenquellen transparent in R so zur Verfügung zu stellen, dass die Datenbeschaffung und Aufbereitung automatisch im Hintergrund passiert.

```
# Load packages
library(elmos)
library(Boruta)
# Get lot data
ll<-lotList(feFilter(device="Device123", days=40))
# Get yield data
ll$mergeYieldWafer(db="ProdDB")
# Get process data, tool as result value
ll$mergeLotHistory(pivotValue="ANLAGE", feedback=T)
# Define result vector
ll$resultVector("LOSS_EL_PRCT")
# Start feature selection
ll$findBoruta(maxRuns=50)
# Plot top 20 features with inportance
eplot(ll$boruta())
```

Skript 4: Ursachenforschung mit Feature Selection als R-Skript

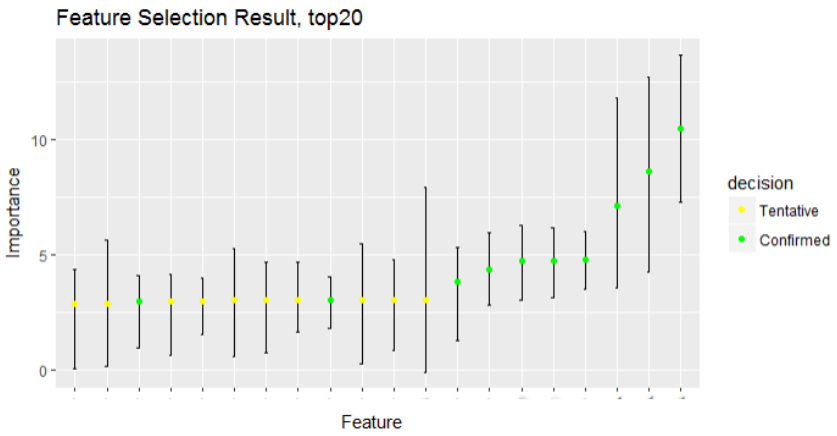

Abb. 6: Resultat der Feature Selection (die Prozessnamen wurden entfernt)

FDC Shotgun

Beim letzten Beispiel aus der Praxis handelt sich um die folgende Fragestellung: Mehrere Wafer einer Charge zeigen Auffälligkeiten bei den Testwerten. Welche Anlage kommt als Verursacher infrage? Dazu verwenden wir FDC Daten (Fault Detection and Classificatian). Dabei handelt es sich um Anlagen-Sensordaten, die als Zeitreihen gesammelt und durch geeignete FDC Methoden prozess-abhängig zu aussagekräftigen Indikatoren verdichtet werden. Am Ende stehen uns pro Wafer und Prozess mehrere Indikatoren zur Verfügung.

Der erste Lösungsansatz zu diesem Problem war, sich die Plots aller Indikatoren der betroffenen Charge anzuschauen und auf mögliche Korrelationen zum Fehlerbild zu prüfen. Leider müssten wir nach einer kurzen Schätzung feststellen, dass es mehr als 10.000 Indikatoren pro Charge gibt. Diese Idee wurde also ziemlich schnell verworfen. Ein R Anwendung musste her.

Mittlerweile ist eine oft benutzte Webanwendung daraus geworden: Die FDC Daten werden per SQL gesammelt und mit dem Fehlerbild, in diesem Falle einfach ein Vektor aus 25 Werten (einer für jeden Wafer einer Charge) korreliert. Ein Pearson- Korrelationskoeffizient für 10.000 Indikatoren ist schnell berechnet und eine Rangliste der Indikatoren - und damit auch der Anlagen die als Ursache infrage kommen - ist in wenigen Sekunden erstell und visualisiert. Auch hier ist das Ergebnis natürlich nur ein Hinweis und muss noch verifiziert werden.

Aufgrund der Methode und der Tatsache, dass in die Gesamtmenge der Indikatoren blind geschossen wird nannte ich die Anwendung – bei allem Pazifismus und Abneigung zu Waffen – „FDC Shotgun“. Tja, ein prägnanter Name ist der halbe Erfolg einer Software…

Fortschritt und Flexibilität brauchen gute Dateninfrastruktur und Full Stack Experten

So mancher fragt sich bestimmt: Wieso treiben wir den Aufwand, machen alles selbst und kaufen nicht fertige Softwarewerkzeuge? Die Antwort ist auf den ersten Blick einfach und dennoch im wahren Unternehmensleben sehr komplex. Zum einen ist unser Industriezweig ein besonderer und dementsprechend speziell sind unsere Anforderungen und Anwendungen. Zum anderen gibt es wenig Wissen über die Halbleiterproduktion in den IT Beraterfirmen, das wir in Anspruch nehmen könnten. Grob gesprochen gibt es zwei Wege, die zu einer Infrastruktur für Datenhaltung und Auswertung führen.

Der erste Weg ist, dass man bereichs- oder problem-spezifische Werkzeuge einkauft, wie z.B. Software zur Verarbeitung und Darstellung der Wafermaps. Auf die Art und Weise erhält man spezialisierte Werkzeuge, die sich sehr gut für die jeweiligen Fragestellungen eignen und schafft anderseits unabhängige Software-Inseln, die bereichsübergreifende Datenauswertung sehr erschweren.

Der andere Weg besteht darin, die Infrastruktur als eine Gesamtheit zu entwerfen und zu erstellen. In diesem Falle erhält man – ein gutes Konzept vorausgesetzt – eine angepasste Struktur mit einfachen bereichsübergreifenden Datenzugriff. Solche Infrastruktur ist allerdings meistens schwer zu pflegen und nach Erreichen ihrer Lebensdauer – wie jede andere Softwareinfrastruktur – nur schwieriger erneuert werden kann.

Die beste Lösung liegt meistens in der Mitte: Zugekaufte Software mit guten Schnittstellen nach außen und selbst erstellte Lösungen überall dort, wo es nichts Passendes auf dem Markt gibt. Das ist der Weg, der bei Elmos seit Jahren verfolgt wird.

Die besten Softwarewerkzeuge sind allerdings nichts wert ohne Leute, die sie bedienen. Data Scientists mit fundiertem Prozesswissen, Erfahrung in Datenverarbeitung, Statistik und Softwareerstellung sind eine gute und notwendige Voraussetzung, um aus den Unternehmensdaten Werte zu schöpfen.

Big Data

Aktuell sind wir bei Elmos dabei, neue Dateninfrastrukturen jenseits der Oracle Landschaft aufzubauen. Eine Cassandra Datenbank mit Spark Cluster soll unsere Engpässe bei der Datenverarbeitung lösen, um noch schneller und effektiver Daten auswerten zu können. Dabei werden wir Stammdaten und Bewegungsdaten wie bisher in der Oracle Datenbank verwalten und diese mit Daten in Big Data Infrastrukturen verknüpfen und auswerten.

Ich hoffe darüber in einem Jahr berichten zu können, Ihr Interesse natürlich vorausgesetzt.

Verweise

1. *Database Model for Zero Defect Quality.* **Peter Czerner, Dr. Ralf Montino.** 2014. DOAG Konferenz.

2. **Wikipedia.** [Online] 2018. https://de.wikipedia.org/wiki/R_(Programmiersprache).

3. **M.B. Kursa, W.R. Rudnicki.** R-Project. [Online] 2018. https://cran.r-project.org/web/packages/Boruta/index.html.

4. **Elmos Semiconductor AG.** Profil. [Online] 2018. https://www.elmos.com/ueber-uns/profil.html.

5. **Leaflet for R**. [Online] RStudio, Inc. https://rstudio.github.io/leaflet/.

6. RStudion Inc. RStudio Home. [Online] 2018. https://www.rstudio.com/.

7. RStudio Inc. Shiny. [Online] RStudion Inc., 2017. https://shiny.rstudio.com/.

8. Wikipedia. Feature Subset Selection. [Online] 2018. https://de.wikipedia.org/wiki/Feature_Subset_Selection.

9. Denis Mukhin, David A. James and Jake Luciani. ROracle. [Online] https://cran.r-project.org/web/packages/ROracle/index.html.

10. R Special Interest Group on Databases (R-SIG-DB), Hadley Wickham, Kirill Müller ORCID iD. DBI. [Online] 2018. https://cran.r-project.org/web/packages/DBI/index.html.

11. Hadley Wickham, Winston Chang , Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo,. ggplot2. [Online] 2018. https://cran.r-project.org/web/packages/ggplot2/index.html.

12. Allround Automations V.O.F. https://www.allroundautomations.com/plsqldev.html. ***PL/SQL Developer 12.0.*** [Online] https://www.allroundautomations.com/plsqldev.html.

Kontaktadresse:

Peter Czerner
Elmos Semiconductor AG
Heinrich-Hertz-Str. 1
44227 Dortmund

Telefon: +49 (0) 231 7549 160
Fax: +49 (0) 231 7549 449
E-Mail peter.czerner@elmos.com
Internet: www.elmos.com