Softwaretechnik

Responsibility-Driven Design

SOFTWARE SYSTEME

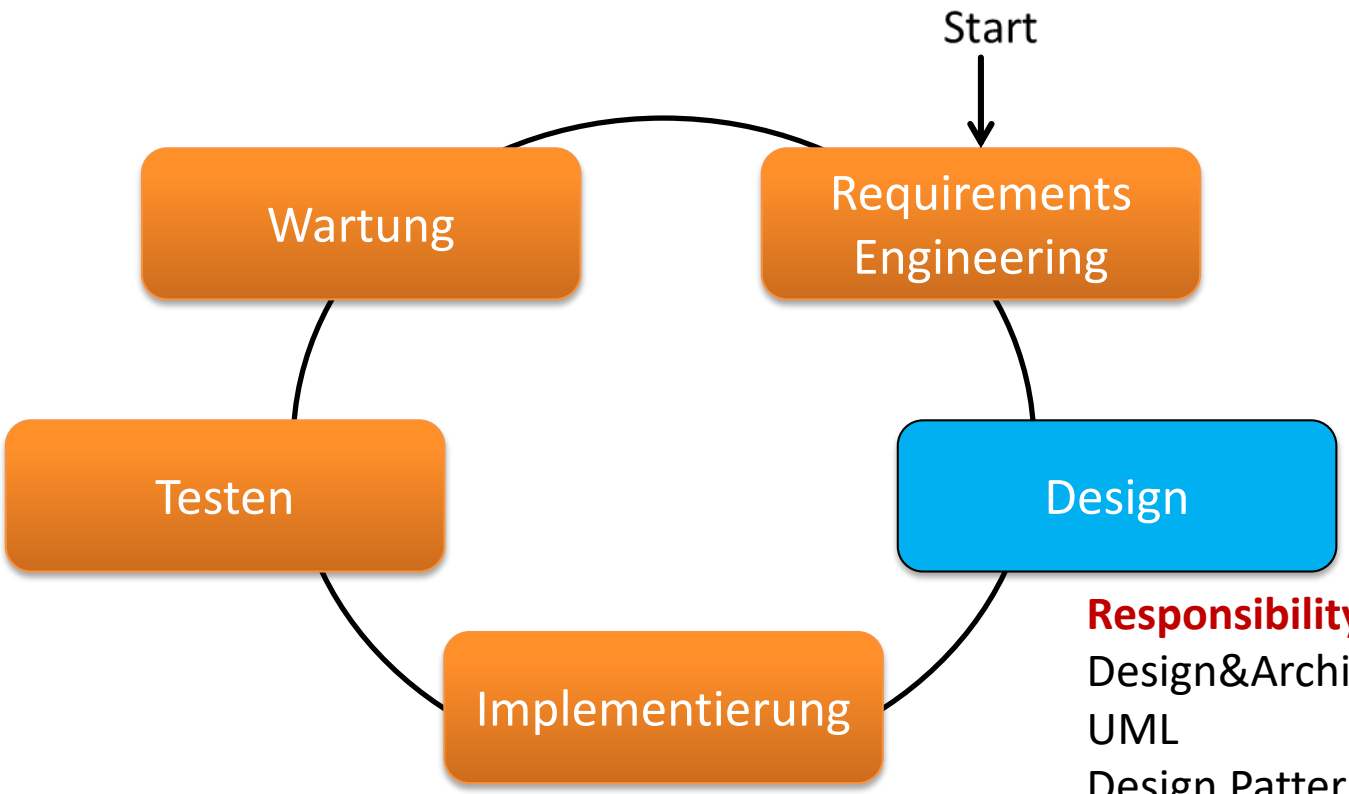
UNIVERSITÄT LEIPZIG

Prof. Dr.-Ing. Norbert Siegmund
Software Systems

Wiederholung: Entwicklungsprozesse

- Herangehensweisen und Prozesse für die Entwicklung von Softwaresystemen
- **Sequentiell:** Strickte Abfolge der Phasen im Lebenszyklus mit Dokumentationen am Ende einer Phase (gut für Projekte mit hohen Sicherheitsanforderungen, klaren und nicht änderbaren Anforderungen)
- **Iterative:** Kurze Entwicklungszyklen, zur Realisierung bestimmter Features in ein auslieferbares Produkt (gut für sich ändernde und unklare Anforderungen, schnelles Feedback, kleinere, agile Teams)
- **Scrum**: Managementframework, welches in festen Sprints Ergebnisse erzielt (gut für neue Produktentwicklung, passt zu agiler Entwicklung; kombinierbar mit Praktiken des XP)
- **Lean**: ähnlich zu Scrum, aber flexibler und darauf ausgelegt, die Arbeit an angefangenen Teilen zu minimieren (gut für Systeme in der Wartung; schnelle Bearbeitung kleinerer Funktionseinheiten)

Einordnung



Responsibility-Driven
Design&Architecture
UML
Design Patterns

Systementwurf und initiales Design

Wie komme ich von Anforderungen / Spezifikationen / User Stories zu einem (objekt-orientierten) Entwurf eines Systems?

Story-driven Design
Entwurfsmethodik mit nicht-IT‘lern

Model-getriebenes Design
Eingebettete Systeme

Responsibility-driven Design
Client-Server Sichtweise im Entwurf / OOP

Data-driven Design
Daten-lastige Systeme

Objektorientierte Analyse und Design
Anwendungen implementiert mit OOP

Feature-orientierte Analyse und Design
Softwareproduktlinien, konfigurierbare Softwaresysteme

Domain-driven Design
Verteilte, multi-purpose Systeme,
Standard für Entwurf von Microservice-Architekturen

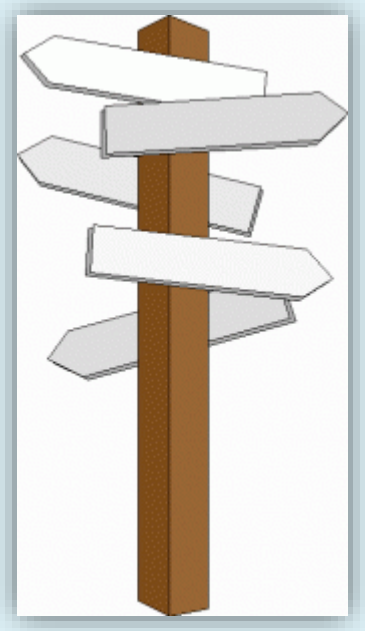
Iteration in Objekt-Orientierten Designs

- Ergebnis des Design-Prozesses ist *kein finales Produkt:*
  - Design-Entscheidung werden evtl. *überdacht*, selbst nach deren Implementierung
  - Design ist nicht linear, sondern *iterativ*

- Der Design-Prozess ist *nicht algorithmisch*:
  - Eine Design-Methode bietet daher nur *Richtlinien* und keine festen Regeln
  - “a good *sense of style* often helps produce clean, elegant designs — designs that make a lot of sense from the engineering standpoint”

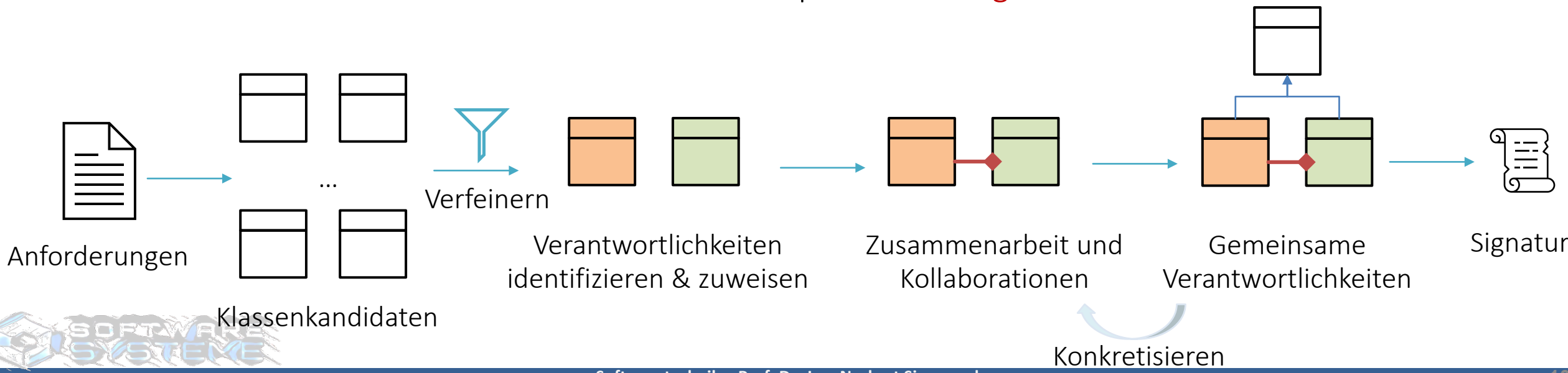
Responsibility-Driven Design ist eine (Analyse- und) Design-Technik, die gut in Kombination mit verschiedenen Methoden und Notationen arbeitet.

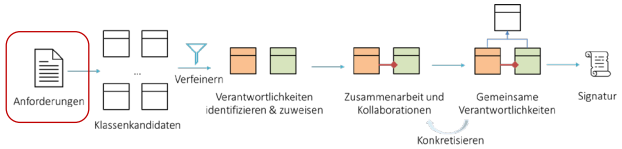
Wie komme ich zu Klassen?



Responsibility-Driven Design: Ablauf

1. Finde die *Klassen* in deinem System (von Kandidaten zu konkreten Klassen)
2. Bestimme die *Verantwortlichkeiten* jeder Klasse
3. Bestimme wie Objekte *zusammenarbeiten,* um ihre Verantwortlichkeiten zu erfüllen
4. Bestimme gemeinsame Verantwortlichkeiten zur Bildung von Klassenhierarchien
5. Konkretisiere Kollaborationen zwischen Objekte
6. Wandle Klassenverantwortlichkeiten in voll spezifizierte Signaturen um



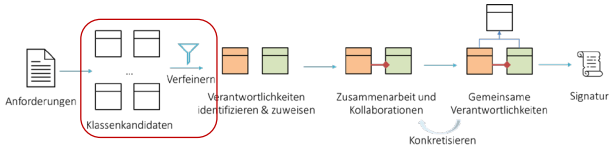


Finden der Klassen

Start mit der Requirements-Spezifikation:

Was sind die Ziele, der erwartete Input und Output des Systems, welches entworfen wird?

1. Suche nach *Nomen Phrasen*:
   - Separiere in offensichtliche Klassen, Kandidaten für Klassen und keine Klassen



Finden der Klassen...

2. Verfeinere die Liste von *Kandidaten*.

Mögliche Hinweise:

- Modelliere *physikalische Objekte* — z.B. Festplatte, Drucker
- Modelliere *konzeptuelle Entitäten* — z.B. Windows, Dateien
- Wähle *ein Wort für ein Konzept* — Was bedeutet das Konzept innerhalb des Systems?
- Vorsicht bei *Adjektiven* — Ist es wirkliche eine eigenständige Klasse?
- Modelliere *Kategorien von Klassen* — Verschiebe noch Modellierung von Vererbungen
- Modelliere *Interfaces* zum System — z.B., Nutzerinterface, Programminterface
- Modelliere Attribut*werte*, nicht Attribute — z.B., Punkt vs. Center

Beispiel: Drawing Editor Requirements Specification

The drawing editor is an interactive graphics editor. With it, users can create and edit drawings composed of lines, rectangles, ellipses and text.

Tools control the mode of operation of the editor. Exactly one tool is active at any given time.

Two kinds of tools exist: the selection tool and creation tools. When the selection tool is active, existing drawing elements can be selected with the cursor. One or more drawing elements can be selected and manipulated; if several drawing elements are selected, they can be manipulated as if they were a single element. Elements that have been selected in this way are referred to as the current selection. The current selection is indicated visually by displaying the control points for the element. Clicking on and dragging a control point modifies the element with which the control point is associated.

When a creation tool is active, the current selection is empty. The cursor changes in different ways according to the specific creation tool, and the user can create an element of the selected kind. After the element is created, the selection tool is made active and the newly created element becomes the current selection.

The text creation tool changes the shape of the cursor to that of an I-beam. The position of the first character of text is determined by where the user clicks the mouse button.

The creation tool is no longer active when the user clicks the mouse button outside the text element. The control points for a text element are the four corners of the region within which the text is formatted. Dragging the control points changes this region. The other creation tools allow the creation of lines, rectangles and ellipses. They change the shape of the cursor to that of a crosshair. The appropriate element starts to be created when the mouse button is pressed, and is completed when the mouse button is released. These two events create the start point and the stop point.

The line creation tool creates a line from the start point to the stop point. These are the control points of a line. Dragging a control point changes the end point.

The rectangle creation tool creates a rectangle such that these points are diagonally opposite corners. These points and the other corners are the control points. Dragging a control point changes the associated corner.

The ellipse creation tool creates an ellipse fitting within the rectangle defined by the two points described above. The major radius is one half the width of the rectangle, and the minor radius is one half the height of the rectangle. The control points are at the corners of the bounding rectangle. Dragging control points changes the associated corner.

Drawing Editor: Nomen Phrasen

The drawing editor is an interactive graphics editor. With it, users can create and edit drawings composed of lines, rectangles, ellipses and text.

Tools control the mode of operation of the editor. Exactly one tool is active at any given time.

Two kinds of tools exist: the selection tool and creation tools. When the selection tool is active, existing drawing elements can be selected with the cursor. One or more drawing elements can be selected and manipulated; if several drawing elements are selected, they can be manipulated as if they were a single element. Elements that have been selected in this way are referred to as the current selection. The current selection is indicated visually by displaying the control points for the element. Clicking on and dragging a control point modifies the element with which the control point is associated.

When a creation tool is active, the current selection is empty. The cursor changes in different ways according to the specific creation tool, and the user can create an element of the selected kind. After the element is created, the selection tool is made active and the newly created element becomes the current selection.

...

Drawing Editor: Nomen Phrasen

The text creation tool changes the shape of the cursor to that of an I-beam. The position of the first character of text is determined by where the user clicks the mouse button. The creation tool is no longer active when the user clicks the mouse button outside the text element. The control points for a text element are the four corners of the region within which the text is formatted. Dragging the control points changes this region. The other creation tools allow the creation of lines, rectangles and ellipses. They change the shape of the cursor to that of a crosshair. The appropriate element starts to be created when the mouse button is pressed, and is completed when the mouse button is released. These two events create the start point and the stop point.

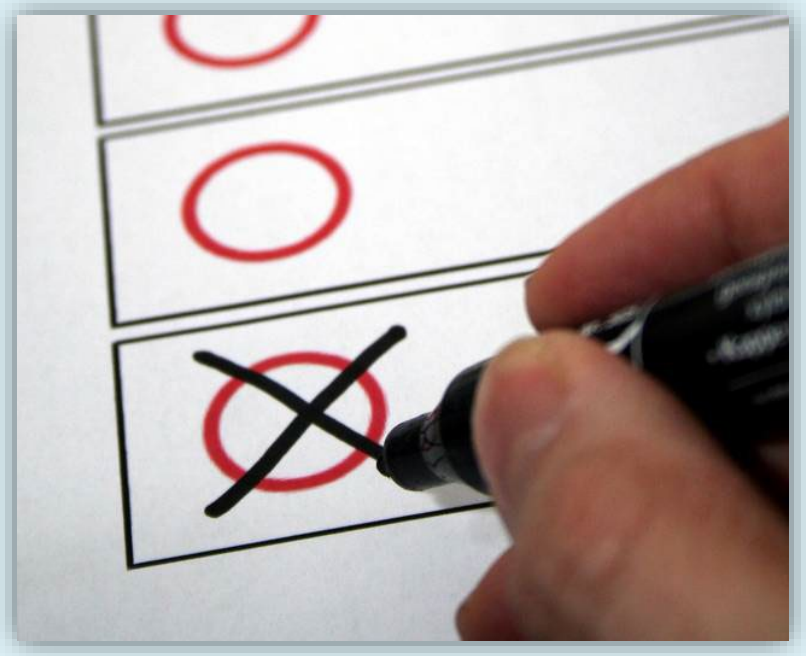
The line creation tool creates a line from the start point to the stop point. These are the control points of a line. Dragging a control point changes the end point.

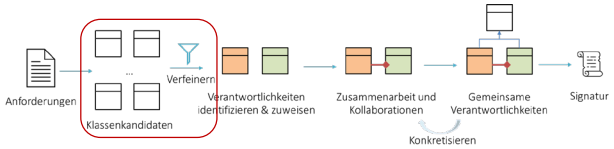
The rectangle creation tool creates a rectangle such that these points are diagonally opposite corners. These points and the other corners are the control points. Dragging a control point changes the associated corner.

The ellipse creation tool creates an ellipse fitting within the rectangle defined by the two points described above. The major radius is one half the width of the rectangle, and the minor radius is one half the height of the rectangle. The control points are at the corners of the bounding rectangle. Dragging control points changes the associated corner.

Wie wähle ich Klassen aus?

- **Physikalische Objekte**
- **Konzeptuelle Entitäten**
- **Ein Wort für ein Konzept**
- **Vorsicht bei Adjektiven**
- **Kategorien von Klassen**
- **Interface zum System**
- **Modelliere Attributwerte**



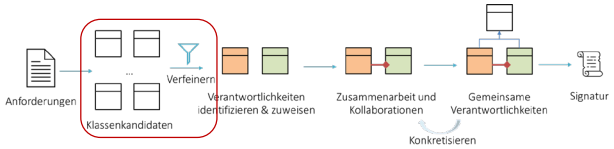


Prinzipen der Klassenauswahl

Modelliere *physikalische Objekte*:

- ~~mouse button~~ [event or attribute]

- Modelliere *konzeptuelle Entitäten*:
  - ellipse, line, rectangle
  - Drawing, Drawing Element
  - Tool, Creation Tool, Ellipse Creation Tool, Line Creation Tool, Rectangle Creation Tool, Selection Tool, Text Creation Tool
  - text, Character
  - Current Selection



Prinzipen der Klassenauswahl …

Wähle *ein Wort für ein Konzept*:

- Drawing Editor

  ⇒ ~~editor, interactive graphics editor~~

- Drawing Element

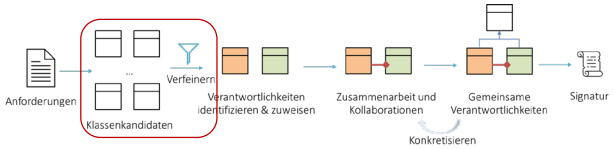
  ⇒ ~~element~~

- Text Element

  ⇒ ~~text~~

- Ellipse Element, Line Element, Rectangle Element

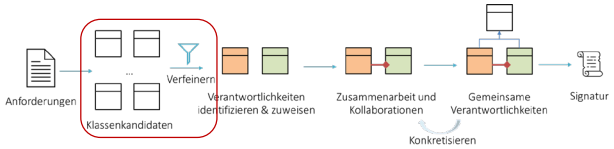
  ⇒ ~~ellipse, line, rectangle~~



Prinzipen der Klassenauswahl …

Vorsicht bei *Adjektiven (in dt. auch zusammengesetzte Nomen)*:

- Ellipse Creation Tool, Line Creation Tool, Rectangle Creation Tool, Selection Tool, Text Creation Tool
  - *Alle haben unterschiedliche Anforderungen*
- Rectangle ⇒ ~~bounding rectangle, rectangle, region~~
  - *Gleiche Bedeutung, aber unterscheiden sich von Rectangle Element*
- Point ⇒ ~~end point, start point, stop point~~
- Control Point
  - *Mehr als nur eine einfach Koordinate*
- Corner ⇒ ~~associated corner, diagonally opposite corner~~
  - *Kein neues Verhalten*



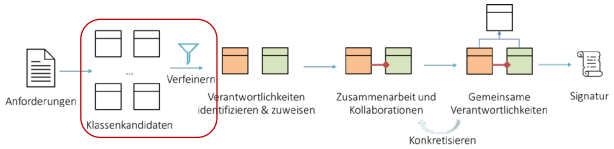
Prinzipen der Klassenauswahl …

Modelliere *Kategorien von Klassen*:

- Tool, Creation Tool

Modelliere *Interfaces* zum System: ***keine guten Kandidaten hier…***

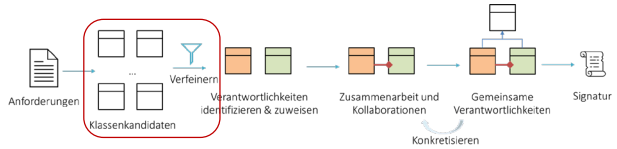
- ~~user~~ — *don't need to model user explicitly*
- ~~cursor~~ — *cursor motion handled by operating system*



Prinzipen der Klassenauswahl …

Modelliere Attribut*werte*, nicht Attribute:

- ~~height of the rectangle, width of the rectangle~~
- ~~major radius, minor radius~~
- ~~position~~ — *des ersten Textzeichens; vermutlich Point-Attribut*
- ~~mode of operation~~ — *Attribut von Drawing Editor*
- ~~shape of the cursor, I-beam, crosshair~~ — *Attribute von Cursor*
- ~~corner~~ — *Attribut von Rectangle*
- ~~time~~ — *Ein implizites Attribut des Systems*



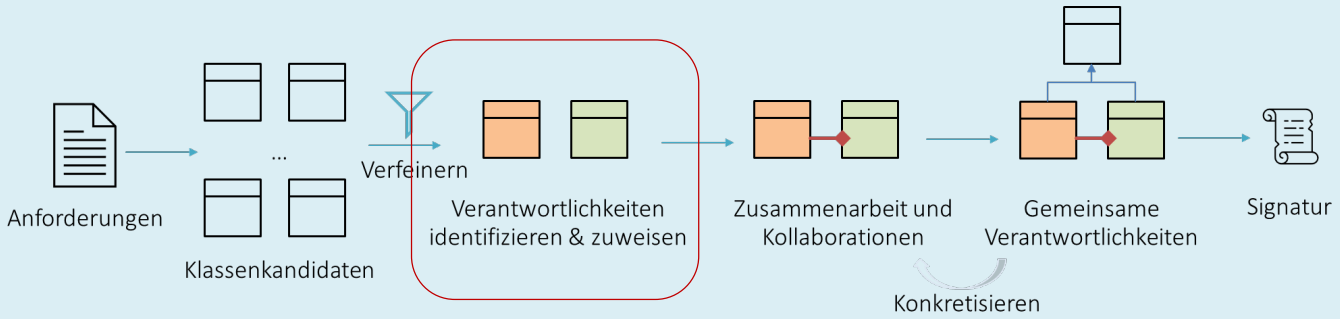
Kandidaten für Klassen

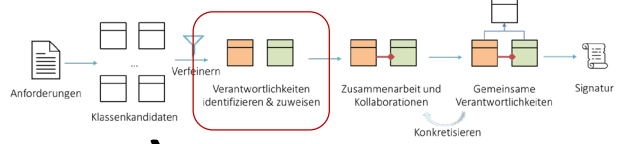
Initiale Analyse ergibt die folgenden Kandidaten:

Character	Line Element
Control Point	Point
Creation Tool	Rectangle
Current Selection	Rectangle Creation Tool
Drawing	Rectangle Element
Drawing Editor	Selection Tool
Drawing Element	Text Creation Tool
Ellipse Creation Tool	Text Element
Ellipse Element	Tool
Line Creation Tool	

Erwartet, dass die Liste sich weiterentwickelt, während Ihr beim Design voranschreitet

Wie identifiziere ich die Verantwortlichkeiten?





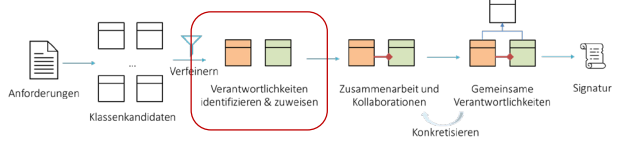
Verantwortlichkeiten (Responsibilities)

Was sind Verantwortlichkeiten?

- Das Wissen, welches ein Objekt verwaltet und anbietet
- Die Aktionen, die es ausführen kann

Verantwortlichkeiten repräsentieren die *öffentlichen Leistungen,* die ein Objekt seinen Klienten anbietet (aber nicht die Art, wie diese Leistungen realisiert werden können)

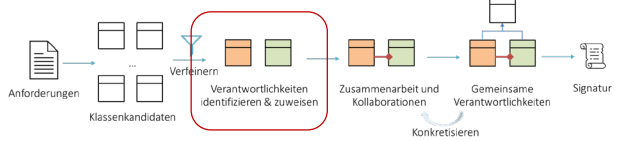
- Spezifiziere *was* ein Objekt tut, nicht *wie* es dies tut
- Beschreibe noch nicht das Interface, sondern nur die *konzeptuellen Verantwortlichkeiten*



Identifizieren der Verantwortlichkeiten

- Studiere die Requirements-Spezifikationen:
  - Hebe *Verben* hervor und bestimme, welche Verantwortlichkeiten diese repräsentieren
  - Mache einen *walk-through* vom System
    - Exploriere so viele Szenarien wie möglich
    - Identifiziere Aktionen, welche durch Eingaben an das System resultieren

- Studiere die Kandidatenklassen:
  - Klassennamen ⇒ Rollen ⇒ Verantwortlichkeiten
  - Aufgenommene „Sinnhaftigkeiten“ von Klassen ⇒ Verantwortlichkeiten

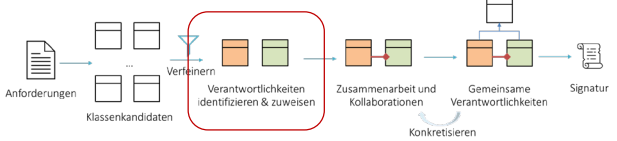


Wie weise ich Verantwortlichkeiten zu?

Pelrine's Laws:

1. "Don't do anything you can push off to someone else."
2. "Don't let anyone else play with you."

- Beispiel: Klasse Buch und Klasse Bibliothek
  - Szenario: Such nach Text in einem Buch
  - Law 1: Bibliothek sollte nicht nach Text in einem bestimmten Buch suchen, obwohl die Klasse eine Liste von Büchern hat.
  - Law 2: Der Text eines Buches gehört zum Buch, also lass niemanden mit deinem Text arbeiten, sondern nur die Klasse Buch bestimmt alle erlaubten Aktionen dafür

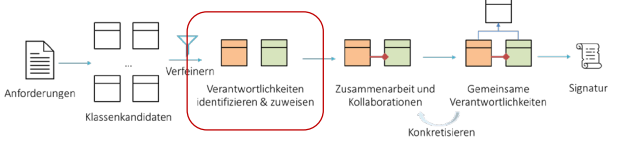


Zuweisen von Verantwortlichkeiten

- *Verteile gleichmäßig* die System-Intelligenz
  - Verhindere prozedural zentralisierte Verantwortlichkeiten
  - Halte Verantwortlichkeiten nah an den Objekten und nicht an deren Nutzern

- Definiere Verantwortlichkeiten so *generell* wie möglich
  - “draw yourself” vs. “draw a line/rectangle etc.”
  - Führt zum Teilen

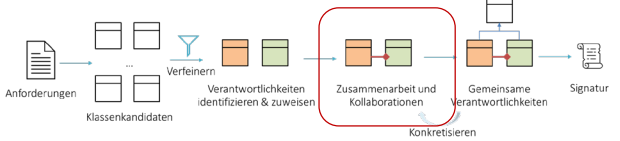
- Halte *Verhalten* zusammen mit jedweder *relevanten Information*
  - Prinzip der Kapselung



Zuweisen von Verantwortlichkeiten…

- Behalte Informationen über eine Sache an *einem Ort*
  - Falls mehrere Objekte Zugriffe auf die gleiche Information benötigen:
    1. Ein neues Objekte kann eingeführt werden, welches die Information verwaltet, oder
    2. Eines der vorhandenen Objekte kann die Information verwalten, oder
    3. die mehrfachen Objekte können in ein einzelnes Objekt überführt werden

- *Teile* Verantwortlichkeiten zwischen ähnlichen Objekten
  - Breche komplexe Verantwortlichkeiten auf

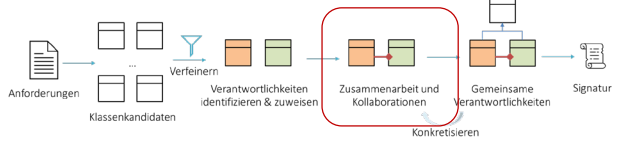


Beziehungen zwischen Klassen

Zusätzliche Verantwortlichkeiten können entdeckt werden, indem wir die Beziehungen zwischen Klassen untersuchen:

- Die “Is-Kind-Of” Beziehung:
  - Klassen, die *gemeinsame Attribute* teilen, teilen oft eine *gemeinsame Oberklasse*
  - Gemeinsame Oberklassen weisen auf *gemeinsame Verantwortlichkeiten* hin
  - z.B., um ein neues Drawing Element zu kreieren, muss das Creation Tool folgendes tun:

1. accept user input — *implemented in subclass*
2. determine location to place it — *generic*
3. instantiate the element — *implemented in subclass*



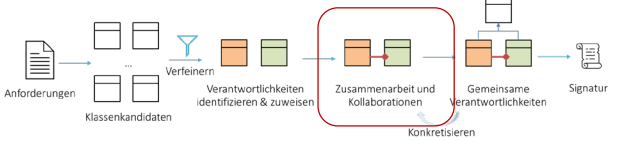
Beziehungen zwischen Klassen ...

- Die “Is-Analogous-To” Beziehung:
  - *Ähnlichkeiten* zwischen Klassen weisen auf eine zur Zeit noch unentdeckte Oberklasse hin

- Die “Is-Part-Of” Beziehung :
  - *Unterscheide* Verantwortlichkeiten zwischen eines *Teils* und des *Ganzen*

Schwierigkeiten bei der Zuweisung von Verantwortlichkeiten weisen auf:

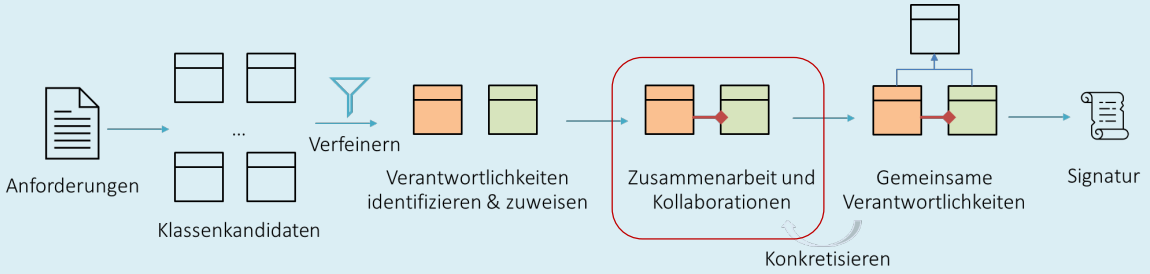
- *Fehlende Klassen* im Design (z.B., Group Element), oder
- *Freie Auswahl* zwischen mehreren Klassen hin

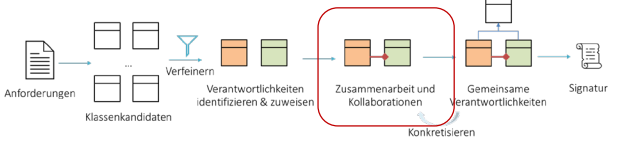


Beispiel Beziehungen

- Drawing element *is-part-of* Drawing
- Drawing Element *has-knowledge-of* Control Points
- Rectangle Tool *is-kind-of* Creation Tool

Wie finde ich Kollaborationen?

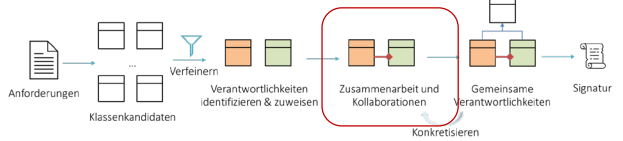




Kollaborationen

Was sind Kollaborationen?

- *Kollaborationen* sind *Benutzerinanfragen (client requests)* an Dienste, die benötigt werden, um Verantwortlichkeiten zu erfüllen
- Kollaborationen enthüllen *Kontroll- und Informationsflüsse* und, ultimativ, Subsysteme
- Kollaborationen können *fehlenden Verantwortlichkeiten* offenbaren
- Analysen von Kommunikationsmustern können *fehlerhaft zugewiesene* Verantwortlichkeiten offenbaren



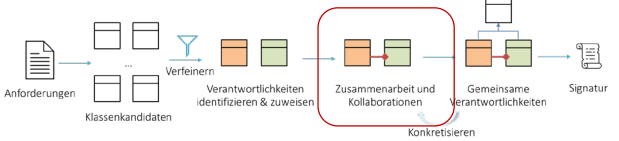
Finden von Kollaborationen

Für jede Verantwortlichkeit:

1. Kann die Klasse die Verantwortlichkeit *selbstständig erfüllen?*
2. Falls nicht, *was benötigt es*, und von welcher anderen Klasse kann es dies erhalten?

Für jede Klasse:

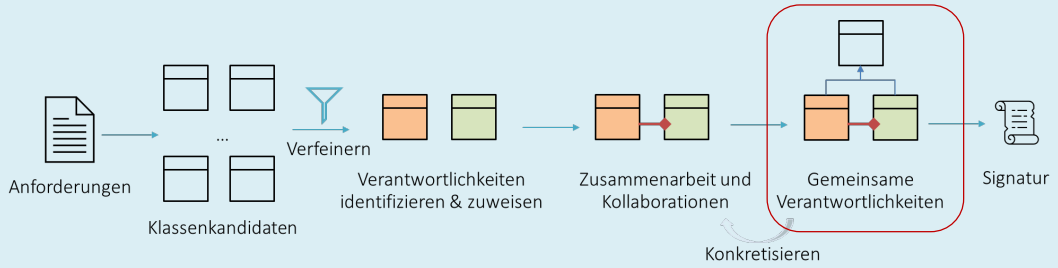
1. Was *weiß* diese Klasse?
2. Welche *anderen Klassen* benötigen ihre Informationen oder Ergebnisse? Prüfe auf Kollaborationen.
3. Klassen, die *nicht* mit anderen *interagieren*, sollten *aussortiert* werden. (sorgfältig prüfen!)

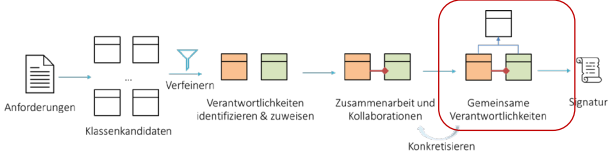


Auflistung der Kollaborationen

Drawing	
Weiß, welche Elemente es verwaltet	
Verwaltet Reihenfolge der Elemente	Drawing Element

Wie finde ich Vererbungshierarchien?

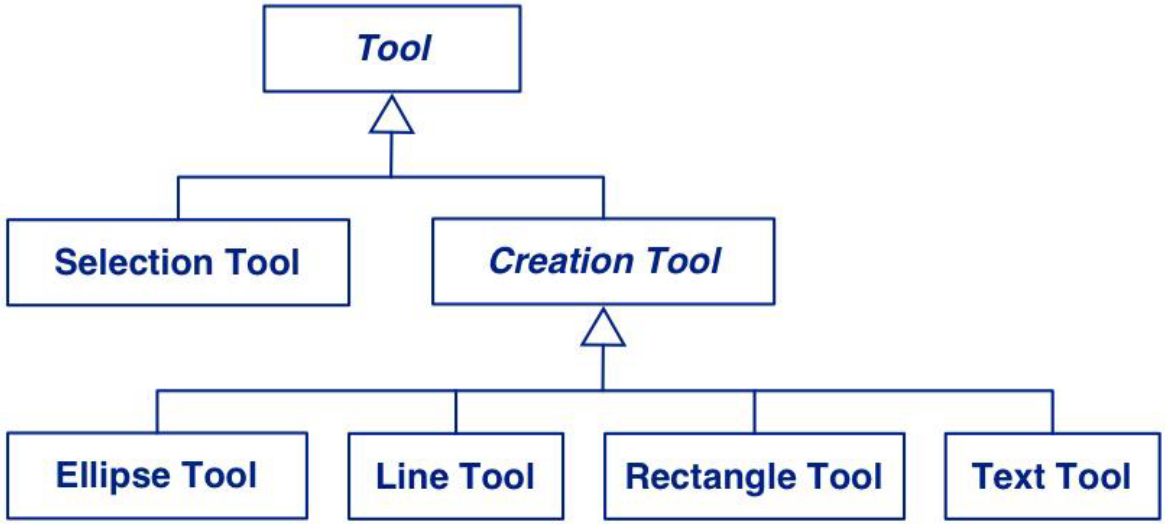


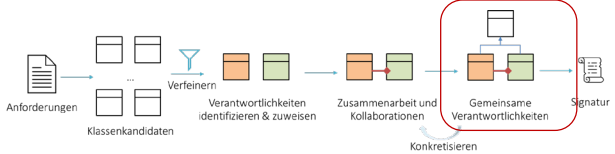


Finden von Abstrakten Klassen

Abstrakte Klassen fassen gemeinsame Verantwortlichkeiten aus anderen Klassen zusammen

- Gruppiere verwandte Klassen mit gemeinsamen Attributen
- Führe abstrakte Oberklassen ein, die diese Gruppe repräsentieren
- “Kategorien” sind gute Kandidaten für abstrakte Klassen

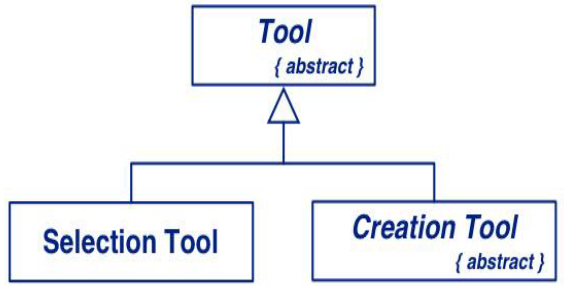




Teilen von Verantwortlichkeiten

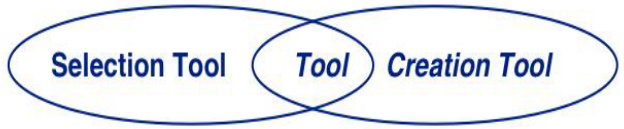
Konkrete Klassen kann man instanziieren und von ihnen erben.

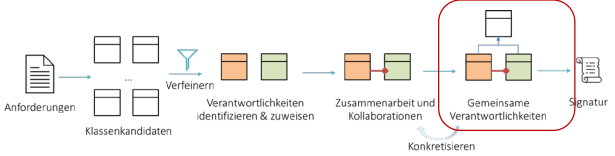
Von abstrakten Klassen kann man nur erben.



Notiere Abstraktheit in Klassendiagrammen.

Venn Diagramme können für die Visualisierung von geteilten Verantwortlichkeiten verwendet werden.



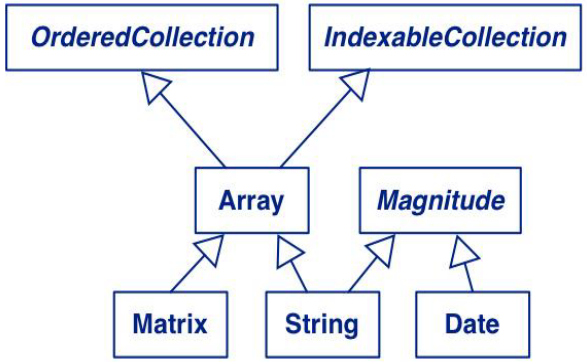


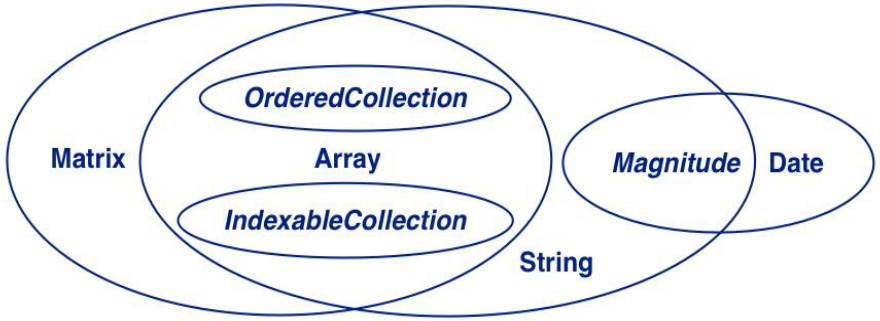
Mehrfachvererbung

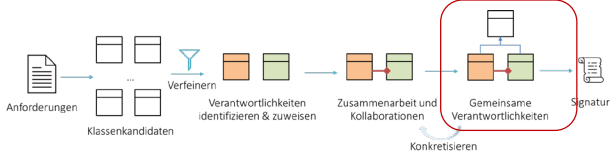
Bestimme, ob eine Klasse *instanziiert* wird, um zu entscheiden, ob sie *abstrakt oder konkret* ist.

Verantwortlichkeiten von Subklassen sind *größer* als diese von *Oberklassen.*

Überschneidungen repräsentieren *gemeinsame Oberklassen.*







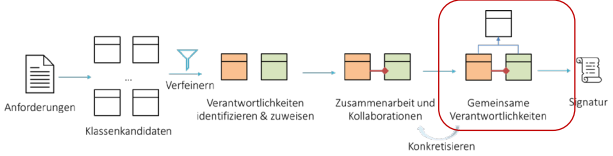
Entwerfen von Guten Hierarchien

Modelliere eine "kind-of" Hierarchie:

- Unterklassen sollten *alle geerbten Verantwortlichkeiten unterstützen*, und eher noch mehr

Schiebe gemeinsame Verantwortlichkeiten so hoch wie möglich:

- Klassen, die *gemeinsame Verantwortlichkeiten teilen* sollten *von einer gemeinsamen abstrakten Superklasse erben*; führe fehlende Superklassen ein



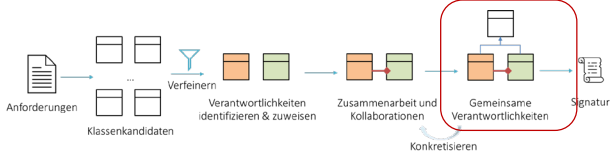
Entwerfen von Guten Hierarchien …

Stelle sicher, dass abstrakte Klassen nicht von konkreten Klassen erben:

- Eliminiere dies durch die Einführung weiterer *gemeinsamer abstrakter Superklassen*: abstrakte Klassen sollten Verantwortlichkeiten in einem implementierungsunabhängigen Weg unterstützen

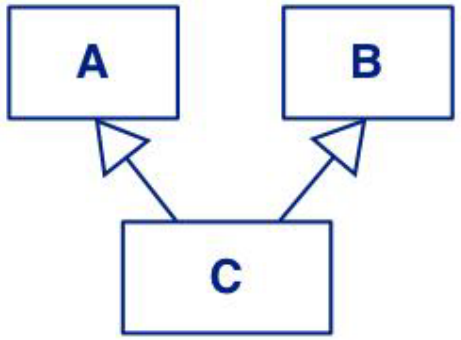
Eliminiere Klassen, die keine neue Funktionalität hinzufügen:

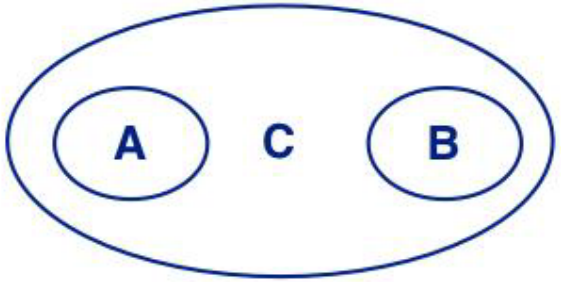
- Klassen sollten entweder neue Verantwortlichkeiten oder eine bestimmte Implementierung von vererbten Verantwortlichkeiten hinzufügen



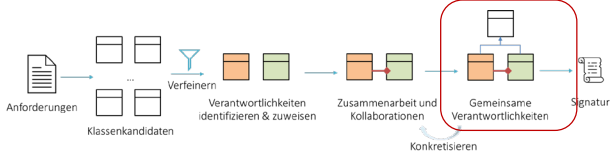
Entwerfen von Kind-Of Hierarchien

Korrekt gebildete Verantwortlichkeiten von Unterklassen:





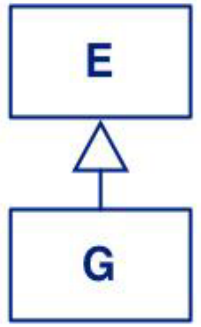
C übernimmt *alle* Verantwortlichkeiten von A und B

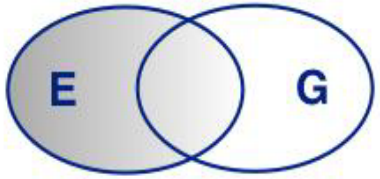


Entwerfen von Kind-Of Hierarchien …

Falsche Unter-Oberklassen-Beziehung

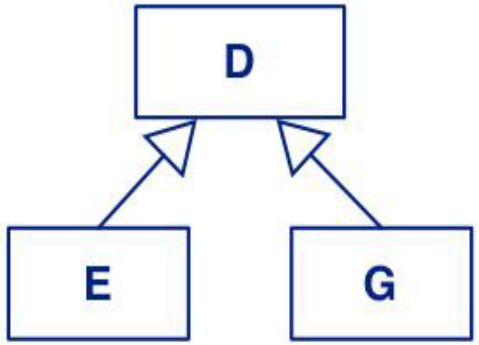
- G übernimmt nur *einige* der Verantwortlichkeiten, welche von E geerbt wurden

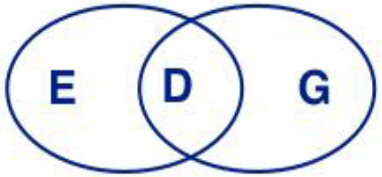


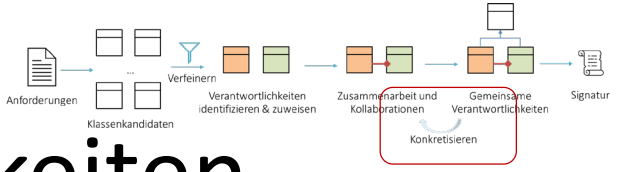


Verfeinerte Vererbungsbeziehung

- Führe eine *abstrakte Oberklasse* ein, welche die gemeinsamen Verantwortlichkeiten kapselt

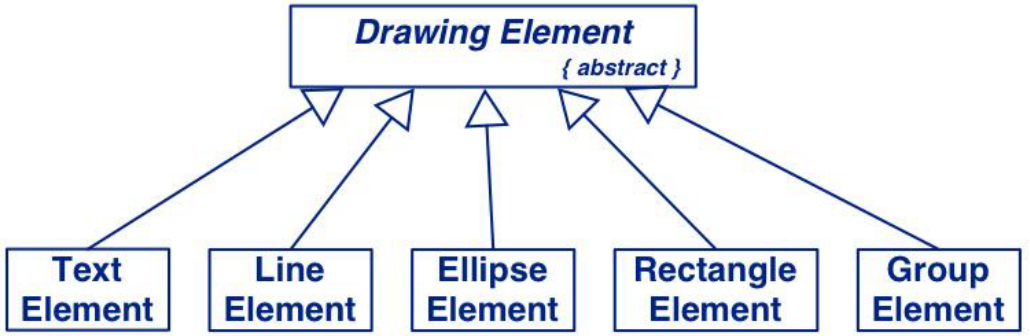




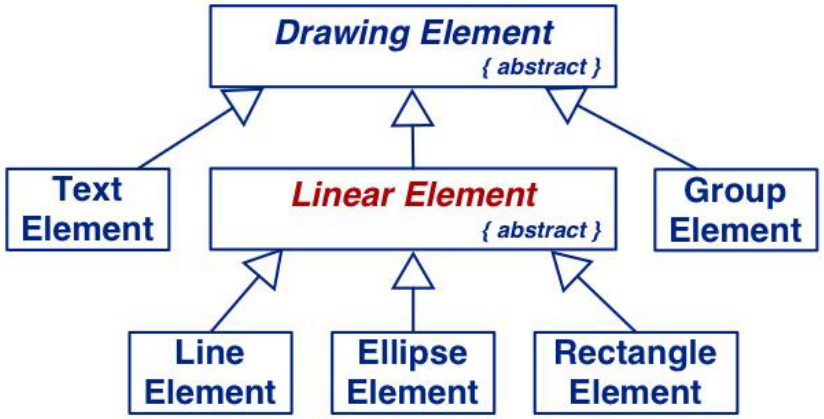


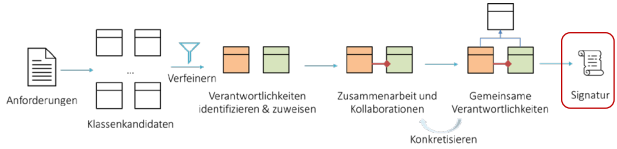
Refaktorisierung von Verantwortlichkeiten

Lines, Ellipses und Rectangles sind verantwortlich, um die Breite und Farbe ihrer Linien zu speichern.



Dies weißt auf eine *gemeinsame Superklasse hin.*





Protokolle (Interfaces)

Ein Protokoll ist eine *Menge von Signaturen* (d.h., ein *Interface*), die zu einer Klasse gehören.

- Protokolle sind für *öffentliche Verantwortlichkeiten* spezifiziert.
- Protokolle für *private* Verantwortlichkeiten sollten spezifiziert werden, falls sie in ihren *Sub(unter)-klassen* benutzt oder implementiert werden (protected Sichtbarkeit)

1. Entwerfe eine Protokoll für jede Klasse
2. Schreibe eine Design-Spezifikation für jede Klasse und Subsystem
3. Schreibe eine Design-Spezifikation für jeden Kontrakt

Was Ihr mitgenommen haben solltet!

- Welche Kriterien gibt es mit denen ich potentielle Klassen identifizieren kann?
- Was sind Verantwortlichkeiten von Klassen und wie kann ich sie identifizieren?
- Wie kann das Identifizieren von Verantwortlichkeiten beim Identifizieren von Klassen helfen?
- Was sind Kollaborationen und wie stehen sie in Beziehung zu Verantwortlichkeiten?
- Wie kann ich abstrakte Klassen identifizieren?
- Welche Kriterien gibt es, um gute Klassenhierarchien zu entwerfen?
- Wie kann das Refaktorisieren von Verantwortlichkeiten Hierarchien verbessern?

Literatur

- *Designing Object-Oriented Software*, R. Wirfs-Brock, B. Wilkerson, L. Wiener, Prentice Hall, 1990.