
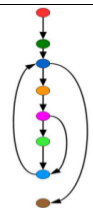
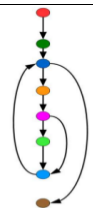
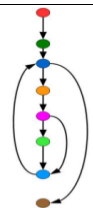


Projektplanung (Inception, Elaboration, Construction, Transition)			Aufwandschätzung			Software Architektur: Architektur ist die Summe der Design-Entscheide, die von grosser Tragweite sind, und die länger leben		
Ablauf	Projektplanung/Requirements → Arbeitspakete, Bugs, SAD → Alles Archivieren und Ergebnis zeigen		Erfolg ist extrem grössenabhängig, vA wegen Scope Creep, mehrere Mitarbeiter, ungenaue Angaben.			Eigenschaften		
Dokumente	SRS(UC, NFR), SAD, Testprotokoll, Benutzerhandbuch, Installationsanleitung, IT Landschaft, Schnittstellen		Brooks Law Adding man power to a late project, makes it later (Einarbeitungszeit 2-3 Monat) → besser splitten und separat Dev			Wiederverwendbar, austauschbar, testbar, einfach verständlich und damit gut dokumentierbar, stabil und langlebig, skalierbar durch versch. Deploy Varianten, Gut aufgeteilt und somit parallel entwickelbar, kein Overengineering		
End of Elaboration (Vendeupunkt)	Kunde wurde verstanden: *Requirements; Scope mit UCs, Domain Model, NF Anforderungen * Entwürfe des GUIs * Architektur Entwurf mit Prototyp * Entwicklungsumgebung * Aufwandschätzung mit Arbeitspaketen		Einflussfaktoren 1. Grösse des Projekts und geforderte Funktionalität, 2. Art und Komplexität, 3. Qualität der Mitarbeiter			Entscheide		
Diagramme	Dienen zur Kommunikation (formaler als Worte), normiert (UML), maximal A3		Kommunikation (n*(n-1) / 2, Requirements, Architecture, Management / Planung, Anforderungsanalyse, Testing			Stateful/stateless; singlethreaded/multithreaded; asynchron/synchron; FIFO/Priority Queue; P2P/Multi P. Broadcast		
Merkatz	"So früh wie möglich so formal wie möglich" "So früh wie möglich so komplett wie möglich"		Nicht lineare Faktoren.			Architektur Analyse		
Inception	Vision, Projekt-Eckwerte, IT-Landschaft, Annahmen&Einschränkungen		Falsche Schätzungen			Umlegende Systeme, Altkoren, Schnittstellen, Randbedingungen (IT Landschaft), Nutzer, Rollen, Rechte, Security, Performance, Usability, Portability, UX Vorgaben		
Elaboration	Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stakeholder Architekturrentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Riskoliste, Zeitaufschreibung		Top Down			Diagramme		
Construction	Deployment Diagram, Datenmodell, Metriken, Test Protokolle		Bottom Up			Context Diagram (Grobe Übersicht der Systeme/Artefakten), Interface Diagram, UC Diagram, Aktivitätsdiagramm, Deployment Diagram, Package Diagram, Zustandsdiagramme, Prozesse/Threads, Message Queues		
Requirements Best Practices (testbar und überprüfbar (min,max,avg))			Cocomo: Constructive Cost Model			Typische Architektur-Muster		
1. Herausfinden der echten Kundewünsche durch kritisches Hinterfragen, Befragungen und Beobachtungen			model that allows one to estimate the cost, effort, and schedule when planning a new SE activity.			Single User Desktop Program		
2. Qualität als (NF-) Requirement aufnehmen (Performance, Scalability, Security, Robustness)			Parameter: Project size, Complexity, Analyst Capability, Programmer Capability, Time Constraints, Fluktuation Personen Monate PM + 3.2 * (KDS) ¹ * 1.05 (KDS: K delivered source instructions, 1000 Zeilen Code)			z.B. Word, Photoshop		
3. Mit Veränderungen umgehen. Software ändert sich ca. 2% pro Monat (Design for Change , Kurze Iterationen)			Entwicklungszeit = 2.5 * PM * 0.38			Multi-Tier Architectures		
Design Best Practices			Vorgehen: 1. Schätzung abgeben, 2. Projekt durchführen, 3. Parameter justieren			Fat Client & Server		
1. DRY : Don't Repeat Yourself : Repetition vermeiden, Keine Code/Documentation Duplication → Gefahr von Inkonsistenzen			Function Points			Mobile = «Fat Client» & Server		
2. Keine Kopplung zwischen konzeptionell unabhängigen Aspekten (Orthogonalität: Verschiedene Objekte sollen unabhängig sein)			External inputs : 4 External interface files: 7 Logical internal tables: 10 External queries: 4			Thin Client: Browser-basiert & Server z.B. Load balanced Web		
3. Hierarchische Zerlegung in Komponenten, Packages. Abhängigkeiten zwischen Komponenten reduzieren. Zyklen vermeiden!			zB 25 Eingabemasken, 5 Interface Files, 15 Reports, 10 external queries, 20 DB-Tabellen			DB-zentrierte Architektur		
4. Design to Test : Testbarkeit hat Einfluss auf die Architektur			75*4+5*7+15*5+10*4+20*10 = 450 Function Points -> in Tabelle & Formeln nachschlagen = 82 PersonenMte			Message-Queue basierte Systeme		
Implementation Best Practices			LOC pro Monat			Producer → Consumer		
1. Fix broken windows : Probleme beheben wenn sie entstehen.			80-150 bei schwierigen Echtzeit Projekten, 300-500 im Schnitt mit guten nicht zu grossen Team, bis ca. 1000 LOC nur mit kleinen Spitzenteams- über 1500 LOC ist unglauwbwürdig (Pfusch, viel Copy/Paste, generiert)			NFR: Nicht Funktionale Anforderungen		
2. Refactor early and often : Bestehenden Code verbessern ohne die Funktionalität zu verändern. Liste von offenen Issues führen.			Ohne Erfahrungswert: 400 LOC pro Monat und Entwickler			Anforderungen an die Qualität (WIE)		
3. Bewusst programmieren : Vermeide Programming by Coincidence. Klares Ziel verfolgen. Annahmen mit Tests dokumentieren.			DLOC			Usability/Accessibility, Availability, Performance, Interoperability, Security, Speicherplatz, Anz. User, Erweiterbarkeit		
Verification Best Practices			Delivered Lines of Code: Ohne Testcode, ohne Prototypen, hohe Qualität des Codes, kein Copy Paste			vorteile		
1. Tests : Früh, häufig und automatisch testen. Code Coverage überwachen. Realistische Testdaten verwenden. Integrationstests			Verlauf des Arbeitsfortschritts (Sigmoid Kurve → 80/20 Regel)			Anforderungen an Architektur, Hardware, Usability/Accessibility ist klar.		
2. Reviews in Sitzungen oder selbständig durchführen. Findings festhalten.			1. Schwieriges wird hinausgeschoben, 2. Funktionen verursachen an anderen Orten neue Probleme, 3. Neue Kundenwünsche kommen auf, 4. Fehleinschätzungen, 5. einfaches stellt sich schwierig heraus, 6. Refactorings			Design By Contract		
Projektmanagement			Reviews			Precondition:		
Technical Debt (Quick and dirty: Long after quick is gone, dirty remains.)			Relative Kosten für Fehlerbehebung			Postcondition:		
Möglichen Konsequenzen schlechter Umsetzung von Software. Den zusätzlichen Aufwand, den man für Änderungen einplanen muss.						Class Invariant		
1. Code-Smells Architektur-Smells 2. fehlende Unit Tests 3. Qualitätsmängel(Performance/Skalierbarkeit, Security Flaws...) 4. "Keine Zeit" → Prozesse werden nicht mehr beachtet (Keine Code Reviews mehr, Schätzung von Issues nicht mehr verifiziert...)			Regeln für Code Reviews:			Beispiel Stack mit Cofoja → Contracts als Spezifikationsmittel gehören zum Interface und sollte daher auch für Interface-Methoden definiert werden.		
Die Bausteine der Programmierer für einen nicht-programmierenden Projektleiter			1. Vorbereiten / Einladen (inkl. Link zu Req. Specs, Docus) 2. Vorbedingungen checken (Guidelines, Code kompiliert?, Doku up to date?) 3. Code Review durchführen (Zeile für Zeile, sachlich bleiben, max. 2-3h) 4. Nacharbeiten 5. Allfällige Nachkontrolle			Behavioral Subtyping (Subtyp muss mindestens Vertrag der Basisklasse erfüllen)		
sichtbar: Backlog, Issue/Bug Tracking, Test-Vorlagen, Test-Protokolle			Wichtige Punkte:			- Subtyp darf Preconditions lockern, aber nicht verschärfen (-Verknüpfung)		
teilweise sichtbar: Build Server Output (Build Fails, Testabdeckung, metrics)			- Verständlichkeit (alle sollten Code verstehen) - Namenswahl (Methoden, Variablen, packages) - Code Smells - Übereinstimmung mit Architektur-Ideen und Diagrammen			- Subtyp darf Postconditions verschärfen aber nicht lockern (&&-Verknüpfung)		
nicht sichtbar: Programmcode, Versionskontrolle, Konfigurations-Dateien			Vorteile			- Subtyp darf Invariante verschärfen aber nicht lockern (&&-Verknüpf.)		
Unsichtbares sichtbar machen			1. Fehler werden früh gefunden → Kosten 2. Wissensaustausch über Architektur 3. Team wird besser (Code)			DBC Zyklus:		
für Kunden und Management: Burn-Down Chart, Story Map mit Farben, Trends von Metriken für Entwickler: zusätzlich Backlog, Metriken, Testabdeckung, SonarQube für IT Infrastruktur: Dashboards mit Ressourcen-Verbrauch (Disk, CPU...), Log-Auswertungen visualisiert (Trends sind besser)			Rollen			1. Declare: Ergänze Interface um Deklaration einer neuen Methode		
Projektautomation			3-5 Personen, Alles protokollieren, Autor abwechseln, 500LoC = ca. 2h, NRF nicht vergessen			2. Refactor: Ergänze alle bestehenden Postconditions um Ausdruck basierend auf dieser neuen Abfrage. Auch Invariant		
CRISP			Requirement Review			3. Contract: Formuliere Pre-/Postcondition für neue Methode		
Build Scripts			Wissens-Transfer vom Kunden zum Entwickler			Pure Method		
GNU Make			Nicht funktionale Anforderungen → Kern-Charakteristiken müssen sichtbar sein - Architektur-Doku und Implementation müssen übereinstimmen - Szenarien durchspielen (z.B. für Erweiterbarkeit)			must not update any preexisting state, but is allowed to modify objects that have been created after entry into the pure method. e.g. map.put(), map.remove() is not pure // „map.contains()", stack.top() sind pure		
Apache Ant			Architektur Review			@ThrowEnsures("ExceptionName", "ensures") method() == old(method()) @Ensures("contains(k) ? result != null : resul == null")		
Maven			Metriken Metriken + Belohnung = Desaster → Reviews wichtig!			Error Handling:		
Continuous Integration:			Projekt besser zu Verstehen, Verbesserungspotential sehen und die Qualität zu erhöhen (Technical Debt besser einschätzen)			Prävention		
Workflow			Künftige Projekte besser abschätzen, Effekt von Anpassungen bewerten, Kosten von Projekten zu sehen			Behandlung		
Vorteile			Indikatoren			Lokale Behandlung wenn möglich oder Fehler nicht weiter relevant, ansonste an Aufrufer delegieren		
Weniger Integrationsprobleme (Merge Conflict, Hardware Probleme), Automatisches Testausführung, Einchecken nur möglich wenn Code ohne Error und Warnungen, Generieren und Darstellen von Metriken (Sonarqube), Bugs werden schneller gefunden, Visualisierung des Projekts mit Metriken, Versionierung von Builds,			Typen			Konservativ: Error Handling Prozedur aufrufen, Fehlermeldung anzeigen, Shutdown		
Best Practice			Tools			Optimistisch: Neutrales Resultat, Bestmögliches plausibles Resultat, Warnung loggen		
Single SRC Repo with Git Flow, Build automatisieren (Maven), Auto. Testing (Unit&Integration), At least daily commits, Ever Commit Builds, Fast build (<10m), Test on clone of Prod, Download latest Exec, Reports, Auto. Deployment			Zugriffe von unterem Layer auf obere Layer			Niemals ein ungenaues Resultat liefern (oft bei kritischen Systemen)		
Features			Zyklometrische Zahl			Robustheit		
C. Delivery			Komplexität Logik von Graph G möglicher Abläufe, liefert Anzahl linear unabhängige Pfade v. Programm			Versuche Software am Laufen zu halten (oft bei unwkritischen Systemen)		
SCRUM			Dynamisch: (Code ausführen) Covertura, EcclEmma, Unit Tests, Integration Tests			Defensive programming		
Beim Continuous Delivery wird die getestete Software in sehr kurzen Zyklen auf den produktiven Server deployed.			Statisch: Checkstyle, FindBugs, ReSharper, STAN, structure101, Metrik Analyse (McCabe), Reviews, Formal Verifikation			Systematische Fehlerprüfung von allen inputs, sowie eine systematische Fehlerbehandlung. Defensives Programmieren macht nur in bestimmten Anwendungsbereichen Sinn, da es mit einem grösseren Aufwand verbunden ist.		
Voraussetzungen			Tangles			Nachteile: u.U nicht testbar, kostet Laufzeit und Programmierzeit, duplizierter code		
Ort, Team deckt alle Fähigkeiten ab, häufige Kommunikation, Kunde im Team/ständig verfügbar. Issues werden eigenverantwortlich zugeordnet, Max 10 Personen Teams			Zugriffe von unterer Layer auf obere Layer			Fehlerbarrikade		
Kein Checkpoint End of Elaboration, "überleben agil" (kein Plan, keine Architektur), höhere Abstraktionsstufe zu User Stories im Backlog fehlt (NFR, Use Cases, Domain Modell etc)			WMC			Exception Policy		
PO (committed)			m = Anzahl Methoden; m/A = Anz. Methoden, die auf ein Attribut zugreifen (Pro Attribut ein Wert)			Exceptions		
PL (involved)			LCOM* = (m - Mittelwert(m(A)))/(m - 1) → Mittelwert m(A) = Anzahl Zugriffe / Anzahl Variablen			Lokales oder globale Behandlung, Checked oderUnchecked Exceptions?		
Sprints			Interpretation: LCOM* = 1 → Schlechte Kohäsion LCOM* = 0 → Maximale Kohäsion			Assertions		
Arbeitspakete			C _e = Affrent Coupling:(Eingehend) Anzahl Klassen ausserhalb eines Packages, die von Klassen innerhalb abhängen			Können ab-, eingeschalten werden, für States die nie eintreten sollen, Pre und Post Conditions		
Story Points			C _a = Effrent Coupling:(Ausgehend) Anzahl Klassen innerhalb eines Packages, die von Klassen ausserhalb abhängen			Logging		
Product Backlog			A = Abstractness: Anzahl abstrakter Klassen (innerhalb eines Packages) → 1 = Viele / 0 = Keine			Für Diagnose, um Fehler zu identifizieren, System Irregularitäten erkennen (Log4j benutzen) → Postmortem Analyse		
Impediments			I = $\frac{C_e}{C_a + C_e}$ → sollte gegen 0 tendieren, (Umsor kleiner I desto stabiler ist der Code → wenig Abhängigkeiten)			Testing		
Story Splitting von Projekten und Arbeitspaketen			Normalized Distance from Main Sequence			Eine hohe Testabdeckung ist eine notwendige aber nicht hinreichende Bedingung. Unit Test beweist nicht die Absenz von Fehler!		
Kein Projekt über eine Million Budget, kein Projekt länger als 9 Monate! (Max 10Pers im Team) → ansonsten splitten nach Kriterien			WMC			Dependency Injection		
Arbeitspakete			Lack of Cohesion of Methods (LOCOM)			1. Interface erstellen 2. Objekte über Konstruktor injection, 3. Fake und Entity implementieren Interface.		
Einzelne Teilbereiche umsetzen			Affrent und Effrent Coupling			Fake		
Konzept-Gruppen einzeln umsetzen			Instability I			Verzweifelte schnelle Implementierung (z.B In-Memory DB)		
geschäfts-Prozessen			Normalized Distance from Main Sequence			Stub		
geografisch			weitere Metriken			Auf Testfall zugeschnittene Antworten		
Einzelne Use-Cases definieren			NOC - Number of Classes, NSC - Number of Children (Anzahl direkter Unterklassen einer Klasse) - Klasse, die Interface implementiert, zählt als Unterklasse des Interfaces, NOI - Number of Interfaces, DIT - Depth of Inheritance Tree (Distanz der Klasse zur Klasse Object in Vererbungshierarchie), NORM - Number of Overridden Methods, NOM - Number of Methods, NOC - Number of Fields, TLOC - Total Lines of Code (zählt: {} ;), Izahl: /b/a, MLOC - Method Lines of Code (Zeilen innerhalb Methoden), Specialization Index: AVG(NORM * DIT / NOM) A - Abstractness (Anzahl abstrakter Klassen / Anzahl Klassen eines Packages)			Mock		
z.B. einzelne Kantone einzeln umsetzen						Mock		
Teillieferungen für Spezielle Rollen (z.B Produktmanager, Einzelkunde, Marketing)			Lack of Cohesion of Methods (LOCOM)			Auf Testfall zugeschnittene Antworten		
Aufteilung von zu grossen Arbeitspaketen (Auflauf im Kleinen) → Erst einmal alles «nice to have» weglassen. (MVP)			Affrent und Effrent Coupling			TDD		
Durchschnitt			Instability I			1. Specify it (Essens, Test, Assert first), 2. Frame it, 3. Evolve it (Simplest first, Refactor mercilessly) → Always baby steps		
Maximum			Normalized Distance from Main Sequence			1. RED: Test fails, Code compiles, No implementation 2. GREEN: Make it passing (Do the simplest thing that could possibly work) 3. Refactor until consistent and good design 4. Integrate in codebase (optional)		
Fallstrikte			weitere Metriken			Typen		
Zu generisch, auch unproduktive Tätigkeiten einplanen (Doku, Server aufsetzen)			NOC - Number of Classes, NSC - Number of Children (Anzahl direkter Unterklassen einer Klasse) - Klasse, die Interface implementiert, zählt als Unterklasse des Interfaces, NOI - Number of Interfaces, DIT - Depth of Inheritance Tree (Distanz der Klasse zur Klasse Object in Vererbungshierarchie), NORM - Number of Overridden Methods, NOM - Number of Methods, NOC - Number of Fields, TLOC - Total Lines of Code (zählt: {} ;), Izahl: /b/a, MLOC - Method Lines of Code (Zeilen innerhalb Methoden), Specialization Index: AVG(NORM * DIT / NOM) A - Abstractness (Anzahl abstrakter Klassen / Anzahl Klassen eines Packages)			Coverage		
sind umsetzungsorientiert, sind sehr kurz, oft kein Kontext, gruppiert in Epics (10-15 US), unterteilt in Subtasks As an (actor), I want (action) so that (business value)						Statement		
As an (actor), I want (action) so that (business value)			NOC - Number of Classes, NSC - Number of Children (Anzahl direkter Unterklassen einer Klasse) - Klasse, die Interface implementiert, zählt als Unterklasse des Interfaces, NOI - Number of Interfaces, DIT - Depth of Inheritance Tree (Distanz der Klasse zur Klasse Object in Vererbungshierarchie), NORM - Number of Overridden Methods, NOM - Number of Methods, NOC - Number of Fields, TLOC - Total Lines of Code (zählt: {} ;), Izahl: /b/a, MLOC - Method Lines of Code (Zeilen innerhalb Methoden), Specialization Index: AVG(NORM * DIT / NOM) A - Abstractness (Anzahl abstrakter Klassen / Anzahl Klassen eines Packages)			Branch		
Bestandteile: ID, Titel, Beschreibung, Akzeptanzkriterien, Story Points, Kunden Prio, gel. Stunden, Status, Kat.						Decision		
Bei Use Cases geht es darum, Interaktionen von Nutzern mit einem System so zu beschreiben, dass das Ziel der Nutzer klar wird. Nutzer können dabei sowohl Menschen als auch andere technische Systeme sein.			NOC - Number of Classes, NSC - Number of Children (Anzahl direkter Unterklassen einer Klasse) - Klasse, die Interface implementiert, zählt als Unterklasse des Interfaces, NOI - Number of Interfaces, DIT - Depth of Inheritance Tree (Distanz der Klasse zur Klasse Object in Vererbungshierarchie), NORM - Number of Overridden Methods, NOM - Number of Methods, NOC - Number of Fields, TLOC - Total Lines of Code (zählt: {} ;), Izahl: /b/a, MLOC - Method Lines of Code (Zeilen innerhalb Methoden), Specialization Index: AVG(NORM * DIT / NOM) A - Abstractness (Anzahl abstrakter Klassen / Anzahl Klassen eines Packages)			Das Verhältnis von ausgewerteten atomaren Werten (Term, Bedingung, ...) innerhalb von Ausdrücken zu allen vorhandenen atomaren Werten in einem Modul.		
sind kundenorientiert ohne Blick auf Implementierung, sind sehr detailliert (fully dressed), bilden oft ganze Geschäftsprozesse ab, Beschreiben den Kontext, widerspiegeln die Benutzersicht. (ca. 8-12 Epics + 1UC)								

