Ablauf Pro Dokumente Grad of Elaboration (Wendepunkt) GUI Diagramme Diele Merksatz "50 Inception V Elaboration V E		Brooks Law Einflussfaktoren Nicht lineare Faktoren. Falsche	rössenabhängig, vA wegen Scope Creep, mehrere Mitarbe Adding man power to a late project, makes it later (Einarb 1. Grösse des Projekts und geforderte Funktionalität, 2. Ar	eitungszeit 2-3 Monat) → besser spliten und separat Dev	Software Architele Eigenschaften Entscheide	Wiederverwendba skalierbar durch ve	t die Summe der Design-Entscheide, die von gro ar, austauschbar, testbar, einfach verständlich u ersch. Deply Varianten, Gut aufgeteilt und somit s; singlethreaded/multithreaded; asynchron/syn	nd damit gut dokumentierbar, stabil und langlebig, t parallel entwickelbar, kein Overengineering	
Dokumente SRS End of Elaboration Mu(wendepunkt) GUI Diagramme Diei Merksatz So Inception V Elaboration L Construction L Construction C Requirements Best Practic 1. Herausfinden der echt C, Qualität als (NF) Requ 3. Mit Veränderungen un Design Best Practics 1. DRY : Don't Repeat Yot 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design 10 Test : Testba	S(UC, NFR), SAD, Testprotokoll, Benutzerhandbuch, Installationsanleitung, IT Landschaft, Schnittstellen nde wurde verstanden: "Requirements; Scope mit UCs, Domain Model, NF Anforderungen * Entwürfe des Jis * Architektur Entwurf mit Prototopy * Entwicklungsungebung * Aufwandschätzung mit Arbeitspaketen einen zur Kommunikation (formaler als Worte), normiert (UML), maximal A3 of rüh wie möglich so formal wie möglich" "So früh wie möglich so komplett wie möglich" Vision, Projekt-Eckwerte, IT-Landschaft, Annahmen&Einschränkungen  Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder Architekturentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung Deployment Diagram, Datenmodell, Metriken, Test Protokolle  lioss (testbar und überprüfbar (min,max,avg.)	Brooks Law Einflussfaktoren Nicht lineare Faktoren. Falsche	Adding man power to a late project, makes it later (Einarbi 1. Grösse des Projekts und geforderte Funktionalität, 2. Ar	eitungszeit 2-3 Monat) → besser spliten und separat Dev	Ĭ.	skalierbar durch v	ersch. Deply Varianten, Gut aufgeteilt und somi	t parallel entwickelbar, kein Overengineering	
End of Elaboration [Wendepunkt] Diagramme Die Diagramme Die Merksatz So Inception Elaboration L Elaboration L Elaboration L Construction E. Requirements Best Practices 1. Ulerausfinden der echt C. Qualität als (NF) Requa 3. Mit Veränderungen un Design Best Practices 1. DRY : Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test : Testba	nde wurde verstanden: "Requirements; Scope mit UCs, Domain Model, NF Anforderungen * Entwürfe des  Ils "Architektur Entwurf mit Prototyp " Entwicklingsumgebung "Aufwandschätzung mit Arbeitspaketen  enen zur Kommunikation (formaler als Worte), normiert (UML), maximal A3  of rüh wie möglich so formal wie möglich" "So früh wie möglich so komplett wie möglich  Vision, Projekt-Eckwerte, IT-Landschaft, Annahmen&Einschränkungen  Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder  Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder  Architekturentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung  Deployment Diagram, Datenmodell, Metriken, Test Protokolle   tioss (testbar und überprüfbar (min,max,avg.)	Einflussfaktoren Nicht lineare Faktoren. Falsche	<ol> <li>Grösse des Projekts und geforderte Funktionalität, 2. Ar</li> </ol>		Entscheide				
(Wendepunkt) GUI Diagramme Die Merksatz "So Inception V Elaboration V Elaboration A Construction E Requirements Best Practi 1. Herausfinden der ech 2. Qualität als (NF-) Requ 3. Mitt Veränderungen un Design Best Practices 1. DRY I Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test: Testbab	Jis * Architektur Entwurf mit Prototyp * Entwicklungsumgebung * Aufwandschätzung mit Arbeitspaketen  enen zur Kommunikation (formaler als Worte), normiert (UML), maximal A3  of rüh wie möglich so formal wie möglich" "So früh wie möglich so komplett wie möglich"  Vision, Projekt-Eckwerte, IT-Landschaft, Annahmen&Einschränkungen  Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder  Architekturentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung  Deployment Diagram, Datenmodell, Metriken, Test Protokolle   Viose (testbar und überprüfbar (min,max,avg)	Nicht lineare Faktoren. Falsche		t did Kompiexitat, 3. Qualitat dei Mitarbeiter					
Diagramme Diei Merksatz So Inception V Elaboration L Construction E Requirements Best Practis 1. Herausfinden der echt 2. Qualität als (NF-) Requ Design Best Practis 1. DRY : Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design 1 Test : Testba	enen zur Kommunikation (formaler als Worte), normiert (UML), maximal A3 of rüh wie möglich so formal wie möglich" "So früh wie möglich so komplett wie möglich" Vision, Projekt-Eckwerte, IT-Landschaft, Annahmen&Einschränkungen Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder Architekturentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung Deployment Diagram, Datenmodell, Metriken, Test Protokolle  lioss (testbar und überprüfbar (min,max,avg.)	Faktoren. Falsche	Kommunikation (n*(n-1) /2. Requirements, Architecture, P	Management / Planung, Anforderungsanalyse, Testing	Architektur		me, Aktoren, Schnittstellen, Randbedingungen (		
Inception V Elaboration L Construction E Requirements Best Practic 1. Herausfinden der echt 2. Qualität als (NF-) Requ 3. Mit Veränderungen un Design Best Practices 1. DRY: Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test: Testba	Vision, Projekt-Eckwerte, IT-Landschaft, Annahmen&Einschränkungen  Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder Architekturentscheidungen, Schichtendlagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung Deployment Diagram, Datenmodell, Metriken, Test Protokolle  dioes (testbar und überprüfbar (min,max,avg)			0 , 0 , 0	Analyse		bility, Portability, UX Vorgaben		
Elaboration L Construction E Requirements Best Practi 1. Herausfinden der als (NF-) Requ 2. Qualität als (NF-) Requ 3. Mit Veränderungen un Design Best Practicus 1. DRY : Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design 10 Test : Testba	Use Cases + Diagramm, Wireframes, NF Anforderungen, Domain Model, Glossar, Liste v. Stackeholder Architekturentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung Deployment Diagram, Datenmodell, Metriken, Test Protokolle lios (testbar und überprifibar (min,max,avg)		politische Vorgaben dominieren ≠ realistische Vorgaben, z		Diagramme			ce Diagramm, UC Diagramm, Aktivitätsdiagramm,	
Construction C Requirements Best Practi 1. Herausfinden der echt 2. Qualität als (NF-) Requa 3. Mit Veränderungen un Design Best Practices 1. DRY: Don't Repeat Yot 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test: Testba	Architekturentscheidungen, Schichtendiagramm, Meilensteine, Testplan, Risikoliste, Zeitaufschreibung  Deployment Diagram, Datenmodell, Metriken, Test Protokolle  tioes (testbar und überprifibar (min,max,avg)	Schätzungen	ermitteln -> Soll/Ist Vergleich			Deployment Diagram, Package Diagram, Zustandsdiagramme, Prozesse/Threads, Message Queues			
Construction  Requirements Best Practi  1. Herausfinden der echt  2. Qualität als (NF-) Requ  3. Mit Veränderungen un  Design Best Practices  1. DRY : Don't Repeat Yot  2. Keine Kopplung zwisch  3. Hierarchische Zerlegur  4. Design to Test : Testba	Deployment Diagram, Datenmodell, Metriken, Test Protokolle tices (testbar und überprüfbar (min,max,avg)	Ton Davin				plsche Architektur-Muster  gle User Desktop Program + UX, Performance, Sicherheit, Verfügbarkeit, geringe System-Komplexität		andreas Contract Manual college	
Requirements Best Practi 1. Herausfinden der echt 2. Qualität als (NF-) Requ 3. Mit Veränderungen un Design Best Practices 1. DRY: Don't Repeat Vot 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test: Testba	tices (testbar und überprüfbar (min,max,avg)	Top Down			z.B. Word, Photo		- Teamfähigkeit, Deployment/Cloud/Work Everywhere, Portierbarkeit		
Herausfinden der echt     Qualität als (NF-) Requ     Mit Veränderungen un     Design Best Practices     DRY: Don't Repeat You     Keine Kopplung zwisch     Hierarchische Zerlegur     Design to Test: Testba		Bottom Up	,		Multi-Tier Architectures		+ Skalierbarkeit & Performance, Verteiltes & koordiniertes Arbeiten		
2. Qualität als (NF-) Requ 3. Mit Veränderungen un Design Best Practices 1. DRY: Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test: Testba	Requirements Best Practices (testbar und überprüfbar (min,max,avg)  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.		-		Performane (asynchron, nicht-deterministische Latenz), Verfügbarkeit, Sicherheit		
3. Mit Veränderungen un Design Best Practices 1. DRY: Don't Repeat You 2. Keine Kopplung zwisch 3. Hierarchische Zerlegur 4. Design to Test: Testba	Qualität als (NF-) Requirement aufnehmen (Performance, Scalability, Security, Robustness)			eter: Project size, Complexity, Analyst Capability, Programmer Capability, Time Constraints, Fluktuation		er	+ User Experience (UX), Performance, Sicherhe		
DRY: Don't Repeat You     Keine Kopplung zwisch     Hierarchische Zerlegur     Design to Test: Testba	3. Mit Veränderungen umgehen. Software ändert sich ca. 2% pro Monat ( <b>Design for Change</b> , Kurze Iterationen)		odel Personen Monate PM = 3.2 * (KDSI) ^ 1.05 (KDSI: K delivered source instructions, 1000 Zeilen Code)		Mobile = «Fat Cli	ent» & Server	Native Apps: +Besseres UX, +mehr Sicherheit; -		
<ol> <li>Keine Kopplung zwisch</li> <li>Hierarchische Zerlegur</li> <li>Design to Test: Testba</li> </ol>	Design Best Practices		Entwicklungszeit = 2.5 * PM ^ 0.38			n Client: Browser-basiert & + Deployment/Cloud/Work Everywhere, Last-Skalierbarkeit, Portabilität - Performance, UX, Browser-Kompatibilität, Sicherheit, Verfügbarkeit, Testb			
<ol> <li>Hierarchische Zerlegur</li> <li>Design to Test: Testba</li> </ol>	ourself : Repetition vermeiden, Keine Code/Documentation Duplication → Gefahr von Inkonsistenzen		Vorgehen: 1. Schätzung abgeben, 2. Projekt durchführen,						
<ol><li>Design to Test: Testba</li></ol>	hen konzeptionell unabhängigen Aspekten (Orthogonalität: Verschiedene Objekte sollen unabhängig sein)	Function Points   External Inputs: 4   External Interface files: 7   Logical Internal tables: 10			DB-zentrierte Arc	chitektur	+ Einfacher Aufbau, Erweiterbarkeit im Kleinen,	. Last-Skallerbarkeit Syndrom, Wartbarkeit, DB-Vendor ausgeliefert	
	ing in Komponenten, Packages. Abhängigkeiten zwischen Komponenten reduzieren. Zyklen vermeiden!	External outputs: 5 External Queries: 4  zB 25 Eingabemasken, 5 interface Files, 15 Reports, 10 external queries, 20 DB-Tabellen			Message-Oueue	lessage-Queue basierte Systeme + Skalierbarkeit, Performance, Separation of Concerns top umsetzbar			
implementation best rite		25*4+5*7+15*5+10*4+20*10 = 450 Function Points -> in Tabelle & Formeln nachschlagen = 82 PersonenMte			Producer → Cons				
			LoC pro Monat 80-150 bei schwierigen Echtzeit Projekten, 300-500 im Schnitt mit guten nicht zu grossen Team, bis ca. 1000 LOC			NFR: Nicht Funktionale Anforderungen Anforderungen an die Qualität (WIE)			
2. Refactor early and often: Bestehenden Code verbessern ohne die Funktionalität zu verändern. Liste von offenen Issues führen.			nur mit kleinen Spitzenteams- über 1500 LOC ist unglaubwürdig (Pfusch, viel Copy/Paste, generiert)			Usability/Accessability, Availability, Performance, Interoperability, Secuirty, Speicherplatz, Anz. User, Erweiterbarkeit			
3. Bewusst programmieren : Vermeide Programming by Coincidence. Klares Ziel verfolgen. Annahmen mit Tests dokumentieren.			Ohne Erfahrungswert: 400 DLOC pro Monat und Entwickler			Vorteile Anforderungen an Architektur, Hardware, Usability/Accessability ist klar.			
Permetton best rectices			Delivered Lines of Code: Ohne Testcode, ohne Prototypen, hohe Qualität des Codes, kein Copy Paste			Design By Contract			
			erlauf des Arbeitsfortschritts (Sigmoid Kurve → 80/20 Regel)			Precondition: Bedingungen, die vor call erfüllt sein müssen (Aufrufer verantwortlich). Prüfe in Methode nie die Precondition			
			Schwieriges wird hinausgeschoben, 2. Funktionen verursachen an anderen Orten neue Probleme, 3. Neue Kundenwünsche			ostcondition: Bedingungen, die nach dem Aufruf erfüllt sein müssen ass Invariant Garantien für Aufrufer, die immer(nach&vor Aufruf) gelten. Verantwortlich dafür ist Implementation der Klasse			
Projektmanagement			ommen auf, 4. Fehleinschätzungen, 5. einfaches stellt sich schwierig heraus, 6. Refactorings						
	and dirty: Long after quick is gone, dirty remains.)	Reviews	retishen best men	Annua .			racts als Spezifikationsmittel gehören zum Interface-Methoden definiert werden.	@Invariant({})	
	en schlechter Umsetzung von Software. Den zusätzlichen Aufwand, den man für Änderungen einplanen muss.	relative Kosten fü	r Fehlerbehebung Regeln für Code Re				nterface-Methoden definiert werden. mindestens Vertrag der Basisklasse erfüllen)	public interface PrimitiveIntStack {     @Requires({"!isFull()"})	
<ol> <li>Code-Smells Architektur-Smells</li> <li>fehlende Unit Tests</li> <li>Qualitätsmängel (Performance/Skalierbarkeit, Security Flaws)</li> <li>"Keine Zeit" → Prozesse werden nicht mehr beachtet (Keine Code Reviews mehr, Schätzung von Issues nicht mehr verifiziert)</li> </ol>			3. Code Review durchführen (Zeile für Zeile, sachlich bleiben, max. 2-3h)				, aber nicht verschärfen (   -Verknüpfung )	@Ensures({ "top() == value", "isEmpty()" })	
<ol> <li>"Keine Zeit" → Prozesse werden nicht mehr beachtet (Keine Code Reviews mehr, Schatzung von Issues nicht mehr verifiziert)</li> <li>Die Baustelle der Programmierer für einen nicht-programmierenden Projektleiter</li> </ol>							ärfen aber nicht lockern( &&-Verknüfung)	void push(int value);	
	Bug Tracking, Test-teilweise sichtbar: Build Server Output nicht sichtbar: Programmcode,	Requirem	ent errors 4. Nacharbeiten	5. Allfällige Nachkontrolle	- Subtyp darf <b>Inva</b>		aber nicht locker ( &&-Verknüpf.)	@Requires({"!isEmpty()"})	
Vorlagen, Test-Protokolle (Build Fails, Testabdeckung, metrics) Versionkontrolle, Konfigurations-Dateien			Wichtigste Punkte:		DbC Zyklus:			@Ensures({"!isFull()", "old (top()) == result" })	
Unsichtbares sichtbar machen			- Verständlichkeit (alle sollten Code verstehen) - Namenswahl (Methoden, Variablen, packages) - Code Smells				klaration einer neuen Methode	int pop();	
	ment: Burn-Down Chart, Story Map mit Farben, Trends von Metriken		() have in a time or one				en Postconditions um Ausdruck basierend auf	@Requires({"!isEmpty()"}) @Ensures({})	
für Entwickler: zusätzlich Backlog, Metriken, Testabdeckung, SonarQube			- Übereinstimmung mit Architektur-Ideen und Diagrammen			dieser neuen Abfrage. Auch Invariant  3. Contract: Formuliere Pre-/Postcondition für neue Method int top(); }			
	hboards mit Ressourcen-Verbrauch (Disk, CPU), Log-Auswertungen visualisiert (Trends sind besser)	Vorteile		nsaustausch über Architektur 3. Team wird besser (Code)				jects that have been created after entry into the	
Projektautomation  CRISP   Complete/	Compile (alle 1'), Unit test (3'), Package/bundle (1h), Integration Test (1h), Deploy (?h-y) /Repeatable/informative/Schedulable/Portable (Flexilibity (z.B. exec only Test), Performance, Extensilibity)	Rollen	Modeartor, Author, Andere Entwickler (Peers), Not				map.put(), map.remove() is not pure // map.co		
		Regeln	3-5 Personen, Alles protokollieren, Author abwechs	- Wichtig für Kostenschätzungen	@ThrowEnsure:	s("ExeptionName",	"ensures") method() == old(method()) @Ens	ures({"contains(k) ? result != null : resul == null"})	
	, plattformabhängig, kein Dep. Mgmt, Datei wird in Targets aufgeteilt, welche parallel laufen können	Requirement Revi Architektur Reviev		teristiken müssen sichtbar sein - Architektur-Doku und	Error Handling				
	, xml-based scripting, portalbe, complex, kaum reuse, viel copy-paste	Alcilitektui keviev	Implementation müssen übereinstimmen - Szenari		Prävention Behandlung		ode Reviews, Unit und Integration Tests, Statisch		
			Metriken + Belohnung = Desaster → Reviews wichtig!				g wenn möglich oder Fehler nicht weiter relevan		
Continuous Integration: Fortlaufendes Erstellen von Software aus mehreren Komponenten (Basis für Continuous Delivery)			Projekt besser zu Verstehen, Verbesserungspotential sehen und die Qualität zu erhöhen (Technical Debt besser einschätzen)				Handling Prozedur aufrufen, Fehlermeldung anz		
Workflow Compile >	• Unit Test > Package > Integration Test > Deploy (Comit early and often, make small changes)		te besser abschätzen, Effekt von Anpassungen bewerten, I				rales Resultat, Bestmögliches plausibles Resulta		
	ntegrationsprobleme (Merge Conflict, Hardware Probleme), Automatisches Testausführung, Einchecken nur		sste SRC-Files (LOC), Cyclomatic Complexiy, Code Guidelin		Robustheit		aues Resultat liefern (oft bei kritischen Systeme am Laufen zu halten (oft bei unkritischen Syste		
möglich wenn wenn Code ohne Error und Warnungen, Generieren und Darstellen von Metriken (Sonarqube), Bugs werden schneller gefunden, Visualisierung des Projekts mit Metriken, Versionierung von Builds,			Konstrukte, Duplicate Code, Kopplung/Kohäsion, Test Coverage, Lesbarkeit, Separation of Concerns, Gutes Naming						
		ypen <u>Umfangsmetriken</u> (Anzahl Codezeilen, Anzahl Unterprogramme), <u>Logische Strukturmetriken</u> (McCabe, WMC), <u>Datenstrukturmetriken</u> (Anzahl Variablen & ihre Gültigkeit), <u>Stilmetriken</u> (Namenskonventionen), <u>Bindungsmetriken</u>			Programming	Systematische Fehlerprüfung von allen Inputs, sowie eine systematische Fehlerbehandlung. Defensive Programmin amming macht nur in bestimmten Anwedungsbereichen Sinn, da es mit einem grösseren Aufwand verbunden ist.			
	C Repo with Git Flow, Build automatisieren (Maven), Auto. Testing (Unit&Integration), At least daily commits, mit Builds, Fast build (<10m), Test on clone of Prod, Download latest Exec, Reports, Auto. Deployment		enstrukturmetriken (Anzani Variabien & inre Guitigkeit) häsion und Kopplung] (Anzahl Packages ausserhalb, von de				nt testbar, kostet Laufzeit und Programmierzeit,		
	nalysis (Metrics, Static analysis, Test coverage, build history) IDE Integration, Review Functionality		namisch: (Code ausführen) Covertura, EclEmma, Unit Tests,		Fehlerbarrikade	Hinter der Barrikad	den sind die Daten gültig. Beim Grenzübertritt ü	perprüfen. (Facade)	
C. Delivery  Beim Continuous Delivery wird die geteste Software in sehr kurzen Zyklen auf den produktiven Server deployed.			Statisch: Checkstyle, FindBugs, ReSharper, STAN, structure101, Metrik Anaylse (McCabe), Reviews, Formal Verification		Exception Policy	, 0 5			
SCRUM		Tangles Zug	riffe von unterem Layer auf obere Layer		Exceptions		duktive Quellen, für sicherheitsrelvante Fehler,		
Voraussetzungen Kun-	nde weiss noch nicht genau was er will, kein klar definierter Scope/eher experimentell, ganzes Team an einem	Zyklomatische Zah	Komplexität Logik von Graph G möglicher Abläufe, liefert	void zaehleZeichen(int &Vokalanzahl,	Assertions		schalten werden, für States die nie eintreten sol		
Ort,	, Team deckt alle Fähigkeiten ab, häufige Kommunikation, Kunde im Team/ständig verfügbar. Issues werden	(McCabe Metrik) (auch Anzahl	Anzahl linear unabhängige Pfade v. Programm	int &GesamtZahl){	Logging		} else if (!a) {} else { assert !b;} → Kein ausführbare		
	eigenverantwortlich zugeordnet, Max 10 Personen Teams		e = Anzahl Kanten (edges), n = Anzahl Knoten (nodes),	cin >> zchn; while(zchn<='A') &&{		rur Diagnose, um F	renier zu identifizieren, System Irregularitäten e	rkennen (Log4J benutzen) → Postmortum Analyse	
	n Checkpoint End of Elaboration, "übertrieben agil" (kein Plan, keine Architektur), höhere Abstraktionsstufe zu	Testfälle für 100% Coverage)	p = Anzahl Komponenten (com <b>p</b> onents) Ganzes Programm: V(G) = e - n + 2p	wniie(zcnn<='A') &&{	Testing Fine hohe Testab	deckung ist eine pr	otwendige aber nicht hinreichende Bedingung. I	Init Test beweise nicht die Absenz von Fehler!	
	er Stories im Backlog fehlt (NFR, Use Cases, Domain Modell etc) nittstelle zum Kunden. Kennt das Buisness		Einzelne Funktion: <b>V(G) = e - n + 2</b>	Gesamtzahl ++;			rstellen,2.Objekte über Konstruktor injection, 3.		
, ,	inittstelle zum Kunden, Kennt das Bulsness Indenkontrolle, IST/SOLL Vergleich, Risiko Beobachtung, Stakeholderanalyse, Verhandelt mit Kunden		Annäherung: $V(G) = nof\ Conditions(if, while, case, for) + 1$	if(zchn=='A'    ){				uf Testfall zugeschnittene Antworten	
	uern 2-3 Wochen. Inkrementelle Ablieferung mit <b>Feedback vom Kuden</b> . Bei jedem Sprint soll der Gesamtwert		Kontrollflussgraph G: Knoten für jede Anweisung, 1	Vokalanzahl ++;			rüft Call Reihenfolge und Inhalt Dummy O		
	den Kunden steigen. → Minimum Viable Product(MVP).		Kante für normale Abläufe, 2 für if, while, for; switch wie				Assert First), 2. Frame it, 3. Evolve it (Simplest fir		
	peitspakete im Sprint Backlog habe eine Grösse von durchschnittlich 2-4 Arbeitstagen. Entwickler schätzen,		if/else if/else, letzte Anweisung in Schlaufe geht zurück zu Schleifenkopf/Condition	cin >> Zchn;			oiles, No implementation 2. GREEN: Make it pas		
Kun	nde priorisiert!		Interpretation: 1-10: Sehr Gut; 11-20: Evtl. verbesserbar,	return;			stent and good design 4. Integrate in codebase (		
	ätzungen auf Studen sind schwierig und basieren oft auf der Erfahrung des schätzenden Entiwkcler.		>20 Verbesserbar, >50 Schlecht	eturi,		Integration Test, S	ystem Tests (Cucumber, Selenium), Performanc	e Test, Usability Test, Blackbox Test, Whitebox Test	
	rry Points sind eine relative Grösse über die Geschwindigkeit des Teams (Erfahrung über das gesamte Team)	WMC	Malaba d Makhada na Class	clomatic Complexity für alle Methoden in Klassen	Coverage		done white father make the first of the second	the stand of the second of the	
	d zu bearbeitende Stories sind detailiert beschrieben	Lack of Cohesion	m = Anzahl Methoden; m(A) = Anz. Methoden, die auf ei				riante, zählt Zeilen, welche durch Tests abgedec		
	dernis, die nicht innerhalb des Scrum Teams beseitigt werden kann. Gründe/Auslöser ausserhalb Team	of Methods	$LCOM^* = (m - Mittelwert(m(A)))/(m - 1)$ $\rightarrow l$	Mittelwert m(A) = Anzahl Zugriffe / Anzahl Variablen			g (if/else, switch,) wird gezählt. (Abgedeckte Zw Isgewerteten atomaren Werten (Term, Bedingu		
Story Splitting von Projekt Kein Projekt über eine M		(LCOM*)	Interpretation: LCOM* = 1 → Schlechte Kohäsion LCO				en Werten in einem Modul.	18,,ciriaio von Ausul uuxen zu allen	
	→ Architektur dem gesamten Team klar und Kunde vollumfänglich verstanden (Erst nach End. Elaboration)	Afferent und		rhalb eines Packages, die von Klassen innerhalb abhängen.				allen möglichen Pfaden in einem Modul. (unreal.)	
	ne Teilbereiche umsetzen Geschäfts-Prozessen Einezlne Use-Cases definieren	Efferent Coupling Vektor (C., C.)	C <sub>e</sub> = Efferent Coupling:(Ausgehend) Anzahl Klassen innerl A= Abstractness: Anzahl abstrakter Klassen (innerhalb ein	halb eines Packages, die von Klassen ausserhalb abhängen.	Integration Testin			(22)	
	pt-Gruppen einzeln umsetzen <b>geografisch</b> z.B einzelne Kantone einzeln umsetzen	Instability	$I = \frac{C_c}{C_c + C_c}$ sollte gegen 0 tendieren. (Umsor kleiner I de				ebiger, +können E2E Test ersetzen, - können ho	ne Laufzeiten haben	
	ferungen für Spezielle Rollen (z.B Produktmanager, Einzelkunde, Marketing)				Unit Testing	•			
	n Arbeitspaketen (Aufteilung im Kleinen) → Erst einmal alles «nice to have» weglassen. (MVP)	Normalized Distance from	stance from (1.1): Zana of Healessners: Backago mit Jautor Abetrakton Klasson		Irreführende Testabdeckung, -oft nichtssagend (2+2=4), -überleben Refactorings nichts, - viel Mocking in oberen Layer				
	rchschnitt was 1 Person in ¼ eines Sprints schafft	Main Sequence	(Interface-Package) werden evt nicht verwendet	A D			orun 2. Get it right 3. Make it fast (nicht zu früh		
· · · · · · · · · · · · · · · · · · ·	aximum was 1 Person in 50%-70% eines Sprints schafft		(0,0): <b>Zone of Pain:</b> Viele konkreten Klassen, mit vielen Al	phängikeiten 📉 📉	Profiling (Whiteb	iox)		(Tool: TimeTracker, Für Java: NetBeans). Jedem	
		Zu generisch, auch unproduktive Tätigkeiten einplanen (Doku, Server aufsetzen) Achsen: Instability, Abstraction		100 / 100	Methoden Aufruf wird ein Timer Feld angehängt und neu kompiliert.  Last-Messungen (Blackbox) Antwort auf: «Wann geht der Server in die Knie?» (Tools: gatling.io, JMeter)				
	d umsetzungsorientiert, sind sehr kurz, oft kein Kontext, gruppiert in Epics (10-15 US), unterteilt in Subtasks		Ziel: Möglichst Nahe bei Main Sequence			.ast-Messungen (Blackbox) Antwort auf: «Wann geht der Server in die Knie?» (Tools: gatling.io, JMeter)  Miederholte Performance Messung Antwort auf: «Sind die typischen Antwortszeiten immer noch etwa gleich?» (Tool: Selenium)			
	an [actor], I want [action] so that [business value]	weitere Metriken	NOC - Number of Classes, NSC - Number of Children (Anz		Application Telen		Viele Sensoren / Log Outputs programmiere		
	standteile: ID, Titel, Beschreibung, Akzeptanzkriterien, Story Points, Kunden Prio, gel. Stunden, Status, Kat.		Interface implementiert, zählt als Unterklasse des Interfa		Usability Testir		etwa zur Mitte des Projektes mit 3 geeigneten Ti	0	
	i Use Cases geht es darum, Interaktionen von Nutzern mit einem System so zu beschreiben, dass das Ziel der Itzer klar wird. Nutzer können dabei sowohl Menschen als auch andere technische Systeme sein.		Inheritance Tree (Distanz der Klasse zur Klasse <i>Object</i> in Vererbungshierarchie), <u>NORM</u> - Number of Overridden				Jser in Usability Lab ("helfen Sie uns", "denken S		
	itzer kiar wird. Nutzer können dabei sowoni menschen als auch andere technische systeme sein.  Id kundenorientiert ohne Blick auf Implementierung, sind sehr detailiert (fully dressed), Bilden oft ganze	ivietious, NON - Number of Methous, NOP - Number of Fields, 1ECC - Total Ellies of Code (zanit. 1,7, szánit. 7/bla							
`			MLOC - Method Lines of Code (Zeilen innerhalb Methoden), <u>Specialization Index</u> : AVG(NORM * DIT / NOM)  A – Abstractness (Anzahl abstrakter Klassen / Anzahl Klassen eines Packages)			"Die Software wird getestet, nicht Sie"), Moderator der Szenario-Aufträge gibt und beobachtet (und fragt, aber nicht hilft).  Varianten mit Paper Prototyping, Clickable Wireframes und Mockups, A/B Testing, UX-Benutzerführung/(Navigation Map)			
Anforderungen) sinc	eschäftsprozesse ab. Beschreiben den Kontext, wiederspiegeln die Benutzersicht. (ca. 8-12 Epics = 1UC)		Δ - Abstractness (Anzahl abstraktor Klasson / Anzahl Vlas		Varianten mit I	Paper Prototyping,	Clickable Wireframes und Mockups, A/B Testing	, UX-Benutzerführung/(Navigation Map)	

