

Advanced Software-Engineering

DHBW Stuttgart
04.11.2025

Janko Dietzsch

„Aktuelles“ – KI-Blase oder nicht???

Why AI Is Not a Bubble*

*The best arguments I've heard on the \$10 trillion question of the moment

DEREK THOMPSON
OCT 15, 2025 · PAID

278 36 40

Share

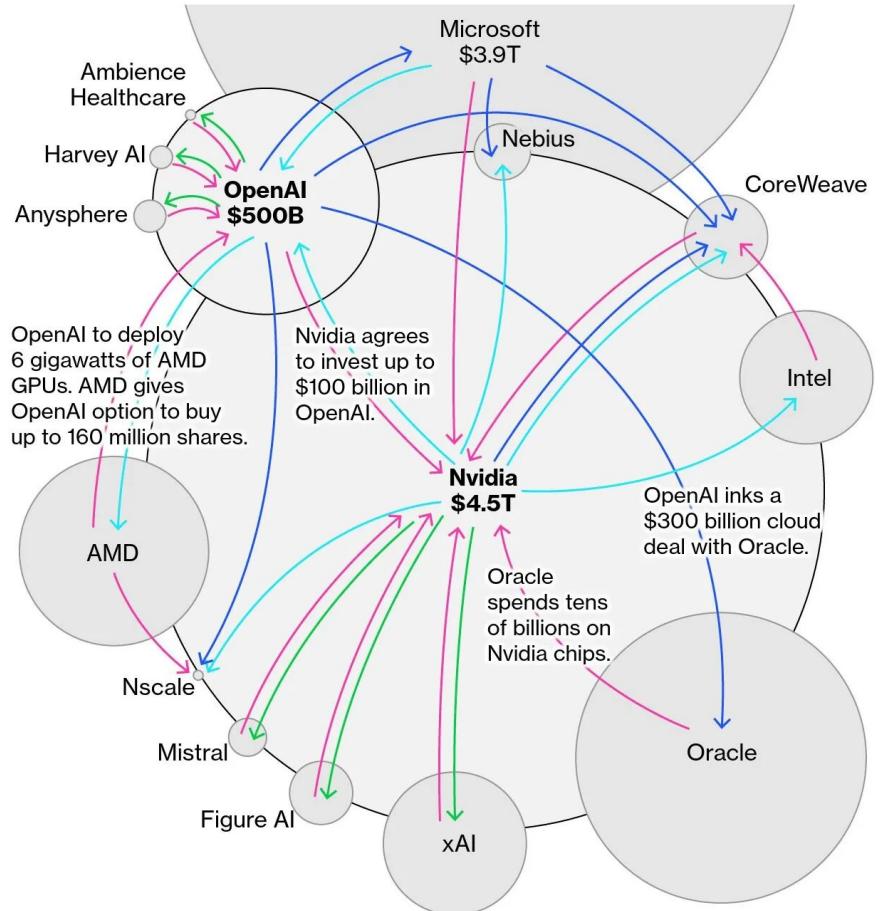
Consensus is a precious commodity these days, so it's striking that every financial newspaper, podcast, newsletter, cable news segment, and hyper-edited TikTok video has erupted in spontaneous agreement that artificial intelligence is history's most obvious bubble. Even the industry titans are all singing the melody line:

- Jeff Bezos called AI "an industrial bubble."
- Bret Taylor, chairman of the board of OpenAI, said, "I think we're also in a bubble."
- David Solomon, CEO of Goldman Sachs, told a recent conference that "there will be a check at some point, there will be a drawdown."
- Jamie Dimon, head of JPMorgan Chase, said he was "far more worried" than others about a big correction ... which is a funny thing to say, because nobody will shut up about how exquisitely aware they are that AI is a bubble.

And yet, look around: Is *anybody* actually acting as if AI is a bubble? Ordinary investors, who are theoretically consuming all this stuff, are still donating their life savings to the silicon gods. (By one estimate, AI companies have accounted for more than 70 percent of US stock gains in 2025.) Bezos's successor, Andy Jassy, has said Amazon will spend over \$100 billion on AI infrastructure this year. OpenAI's plans reportedly imply trillions of dollars in capital spending across the decade.

How Nvidia and OpenAI Fuel the AI Money Machine

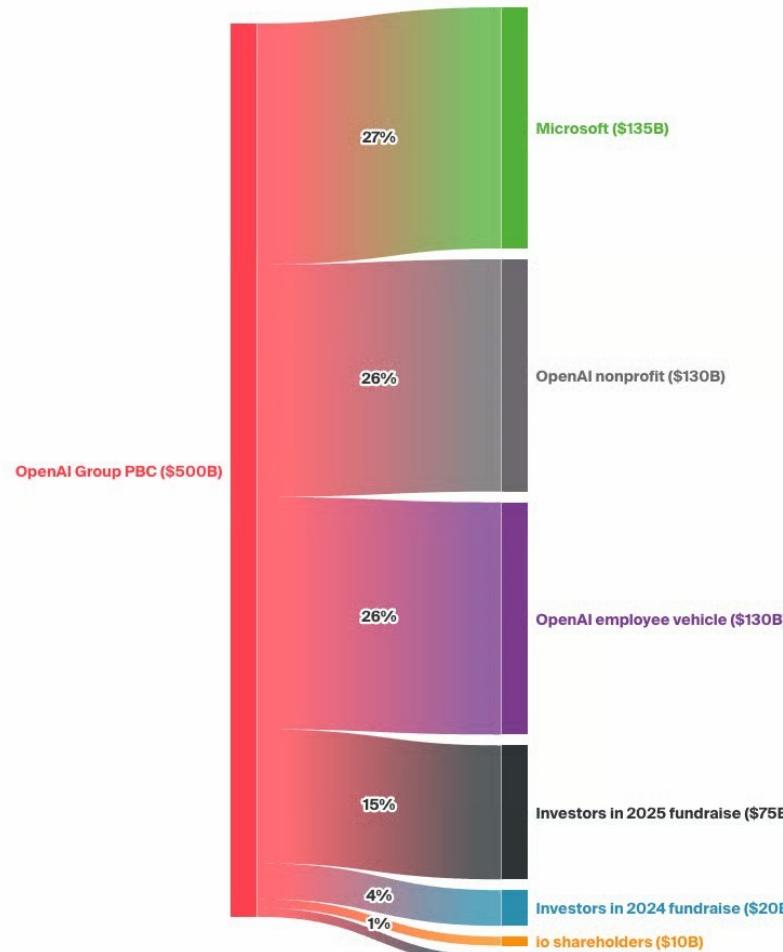
Hardware or Software / Investment / Services / Venture Capital
Circles sized by market value



Source: Bloomberg News reporting

Bloomberg

„Aktuelles“ – 500 Mrd. \$ Restrukturierung bei OpenAI



OpenAI just solved its capital problem by creating a governance paradox: a nonprofit foundation now holds a \$130 billion stake in the for-profit company it oversees, giving it every financial incentive to maximize the value of the AI it's meant to regulate.

The ChatGPT maker [completed its recapitalization](#) on Tuesday, converting its hybrid structure into a public benefit corporation valued at \$500 billion. CEO Sam Altman told employees an IPO is likely and said in a [public livestream](#) it's probable "given the capital needs we will have." Those needs include \$115 billion in projected spending through 2029 and \$1.4 trillion in data center commitments.

The new ownership breakdown:

- Microsoft holds [27% valued at \\$135 billion](#) with IP rights through 2032, including post-AGI models
- OpenAI Foundation controls the board and holds 26% worth \$130 billion, plus warrants if the valuation hits \$5 trillion
- Current and former employees collectively hold 26%
- Recent investors, including SoftBank, hold 15% worth \$75 billion
- 2024 investors, including Thrive Capital, hold 4% worth \$20 billion
- **CEO Sam Altman holds 0%**

Source: OpenAI announcements, The Information reporting • Percentages don't add to 100% because of rounding

„Aktuelles“ – OpenAI – Jährlicher Zubau von 100 GW Leistung nötig

TECH

OpenAI says U.S. needs more power to stay ahead of China in AI: ‘Electrons are the new oil’

PUBLISHED MON, OCT 27 2025 4:38 PM EDT

Ashley Capoot
@IN/ASHLEY-CAPOOT/

SHARE f X in e

KEY POINTS

- OpenAI urged the White House to substantially increase the U.S. investment in new energy capacity to stay ahead of China in AI.
- The startup has been planning ambitious infrastructure buildouts that will require massive amounts of power. It's unclear where all the electricity will come from.
- OpenAI encouraged the U.S. to commit to building 100 gigawatts of new energy capacity each year.



TRENDII



OpenAI möchte die US-Administration überzeugen, dass ein jährlicher Aus- bzw. Zubau von 100 GW für KI nötig ist:

- Verfügbarkeit von viel Energie strategisch für den KI-Ausbau und das Betreiben nötig – insbesondere im Wettbewerb mit China
- 10 GW entspricht dem ungefähren Bedarf von 8 Millionen Haushalten
- Angeblich baute China laut OpenAI letztes Jahr 429 GW auf und die USA nur 51 GW
- **"Electrons are the new oil"**

„Aktuelles“ – They did it again: Modell MiniMax M2

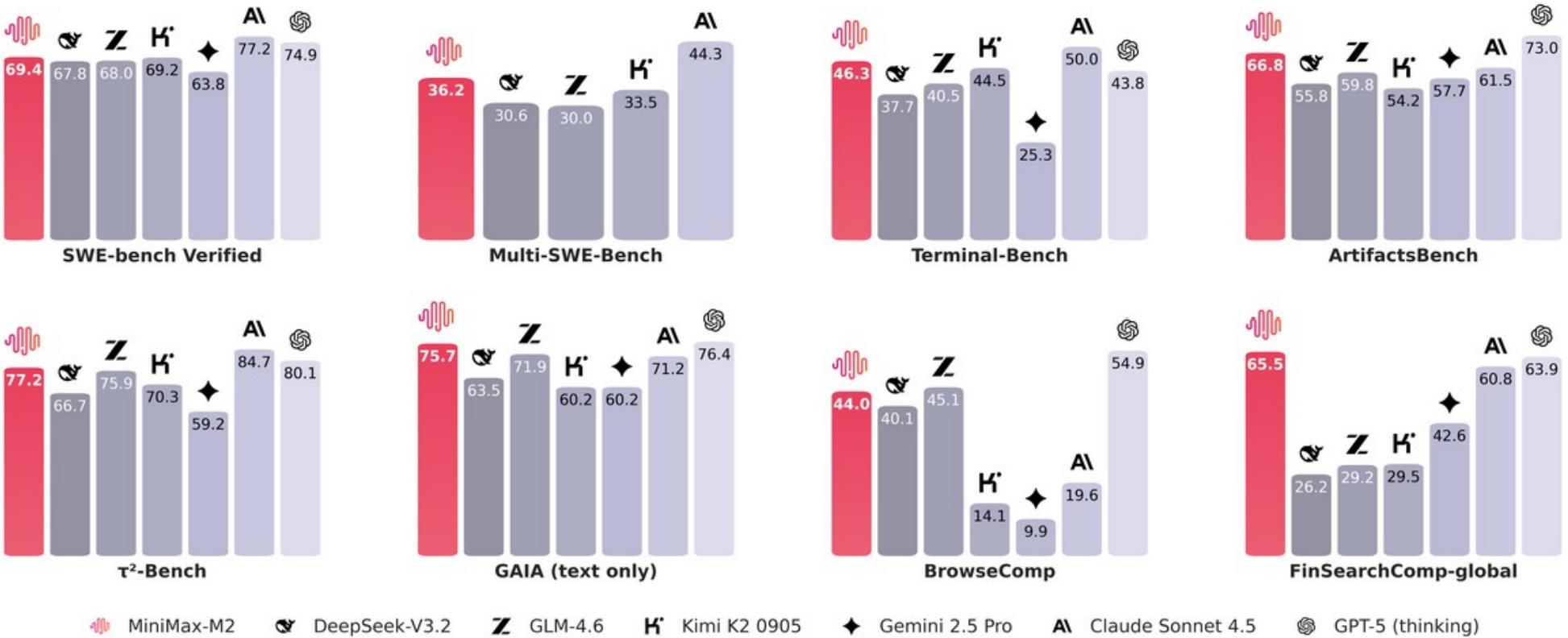


From Day 1 of our founding, we have been committed to the vision of "Intelligence with Everyone."

Today, we are officially open-sourcing and launching MiniMax M2, a model born for Agents and code. At only 8% of the price of Claude Sonnet and twice the speed, it's available for free for a limited time!

- Top-tier Coding Capabilities: Built for end-to-end development workflows, it excels in various applications such as Claude Code, Cursor, Cline, Kilo Code, and Droid.
- Powerful Agentic Performance: It demonstrates outstanding planning and stable execution of complex, long-chain tool-calling tasks, coordinating calls to the Shell, Browser, Python code interpreter, and various MCP tools.
- Ultimate Cost-Effectiveness & Speed: Through efficient design of activated parameters, we have achieved the optimal balance of intelligence, speed, and cost.

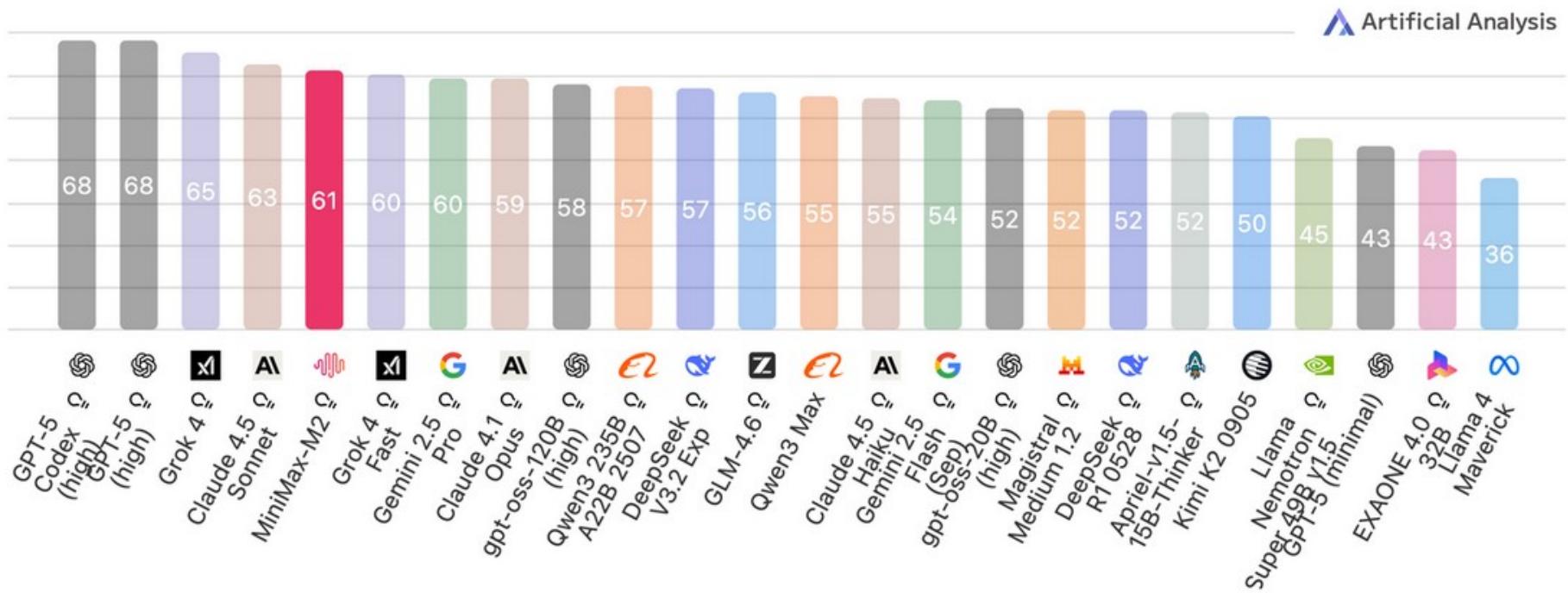
„Aktuelles“ – They did it again: Modell MiniMax M2



„Aktuelles“ – They did it again: Modell MiniMax M2

Artificial Analysis Intelligence Index

Artificial Analysis Intelligence Index v3.0 incorporates 10 evaluations: MMLU-Pro, GPQA Diamond, Humanity's Last Exam, LiveCodeBench, SciCode, AIME 2025, IFFBench, AA-LCR, Terminal-Bench Hard, τ^2 -Bench Telecom

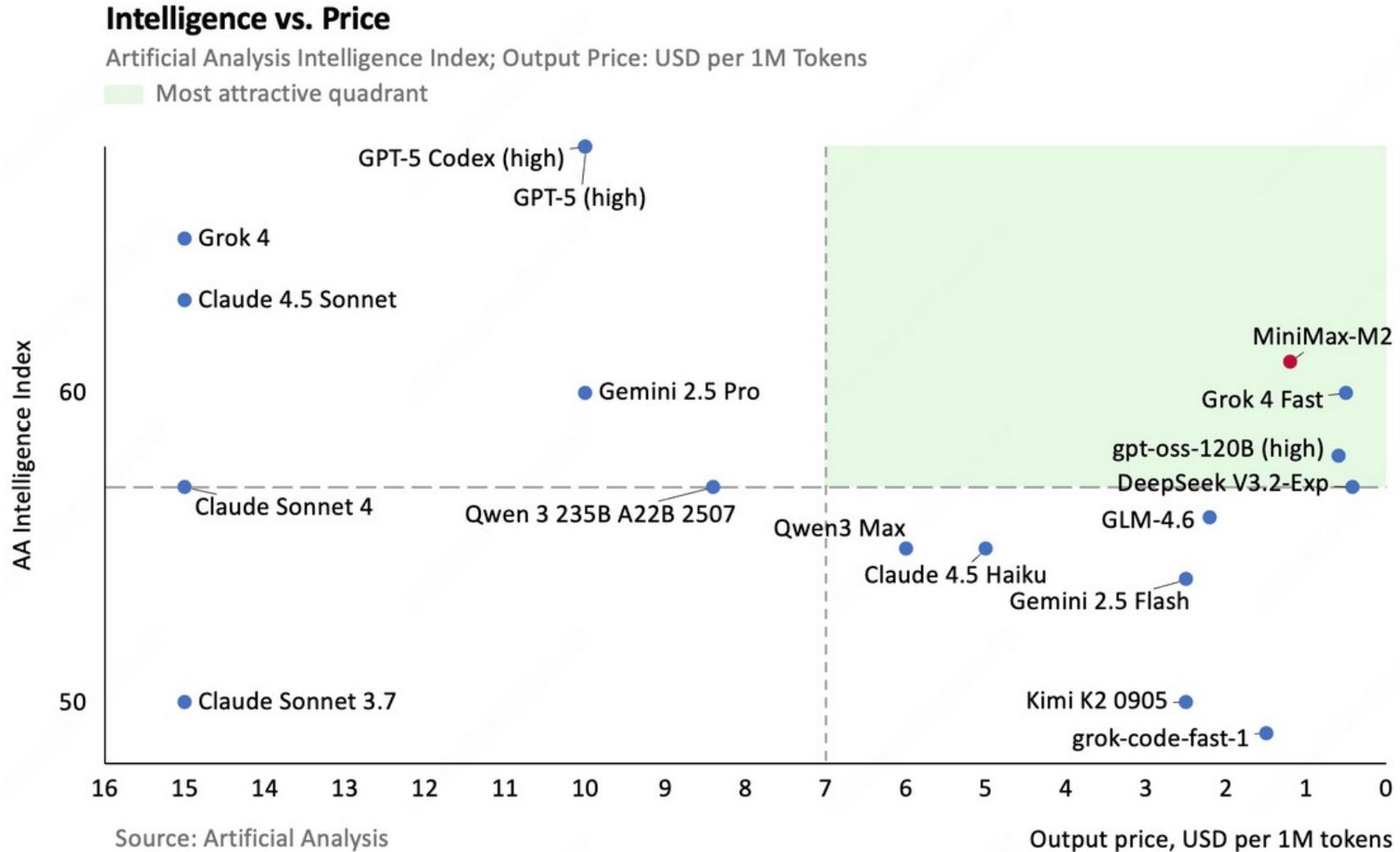


„Aktuelles“ – They did it again: Modell Minimax M2

We have set the API price for the model at \$0.30/¥2.1 RMB per million input tokens and \$1.20/¥8.4 RMB per million output tokens, while providing an online inference service with a TPS (tokens per second) of around 100 (and rapidly improving). This price is 8% of Claude 3.5 Sonnet's, with nearly double the inference speed.

Over the past weekend, many enthusiastic developers from home and abroad have conducted extensive tests with us. To make it easier for everyone to explore the model's capabilities, we are extending the free trial period to November 7th, 00:00 UTC. At the same time, we have **open-sourced the complete model weights** on Hugging Face. Interested developers can deploy it themselves, with support already available from SGLang and vLLM.

„Aktuelles“ – They did it again: Modell MiniMax M2

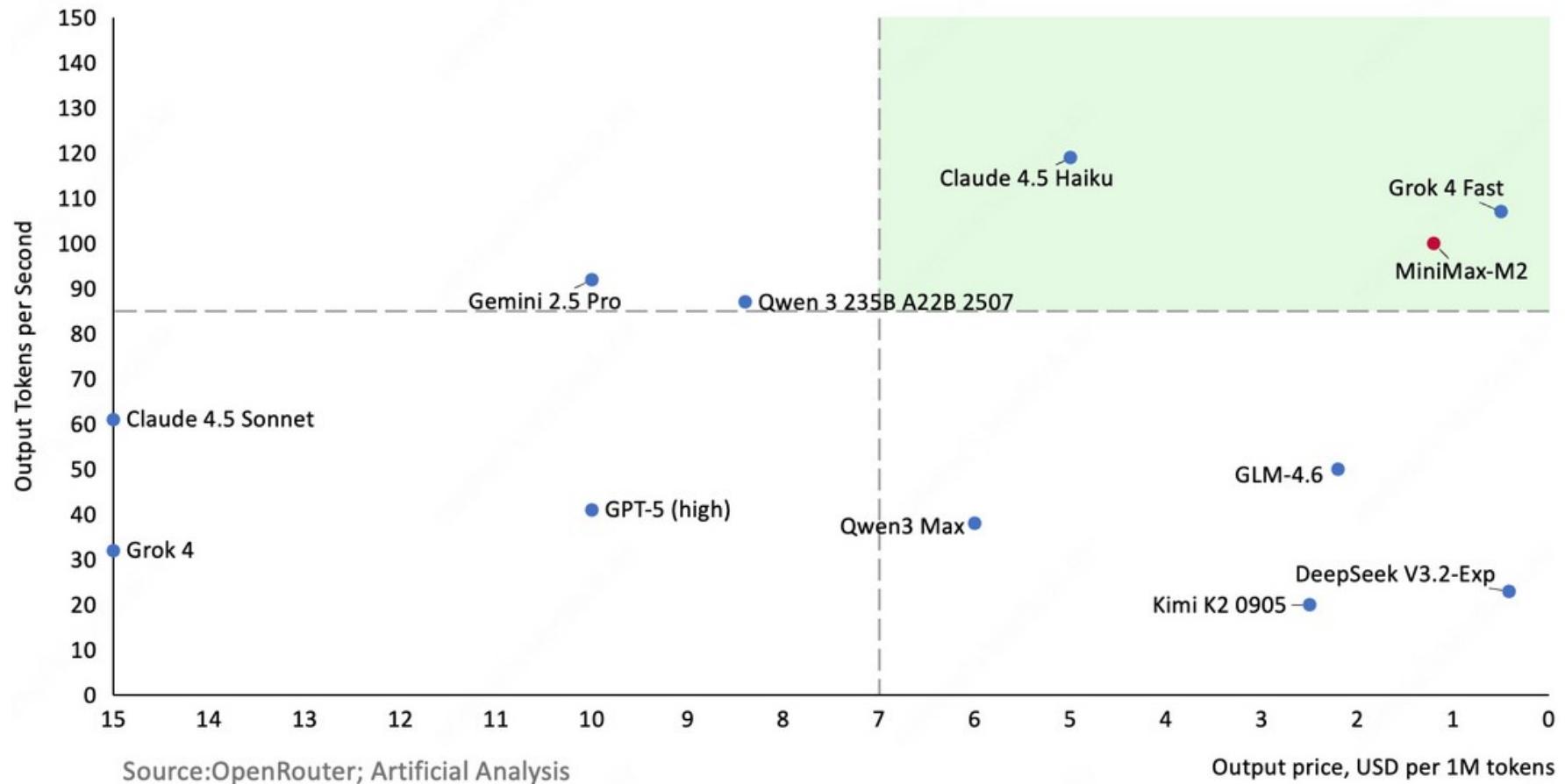


„Aktuelles“ – They did it again: Modell MiniMax M2

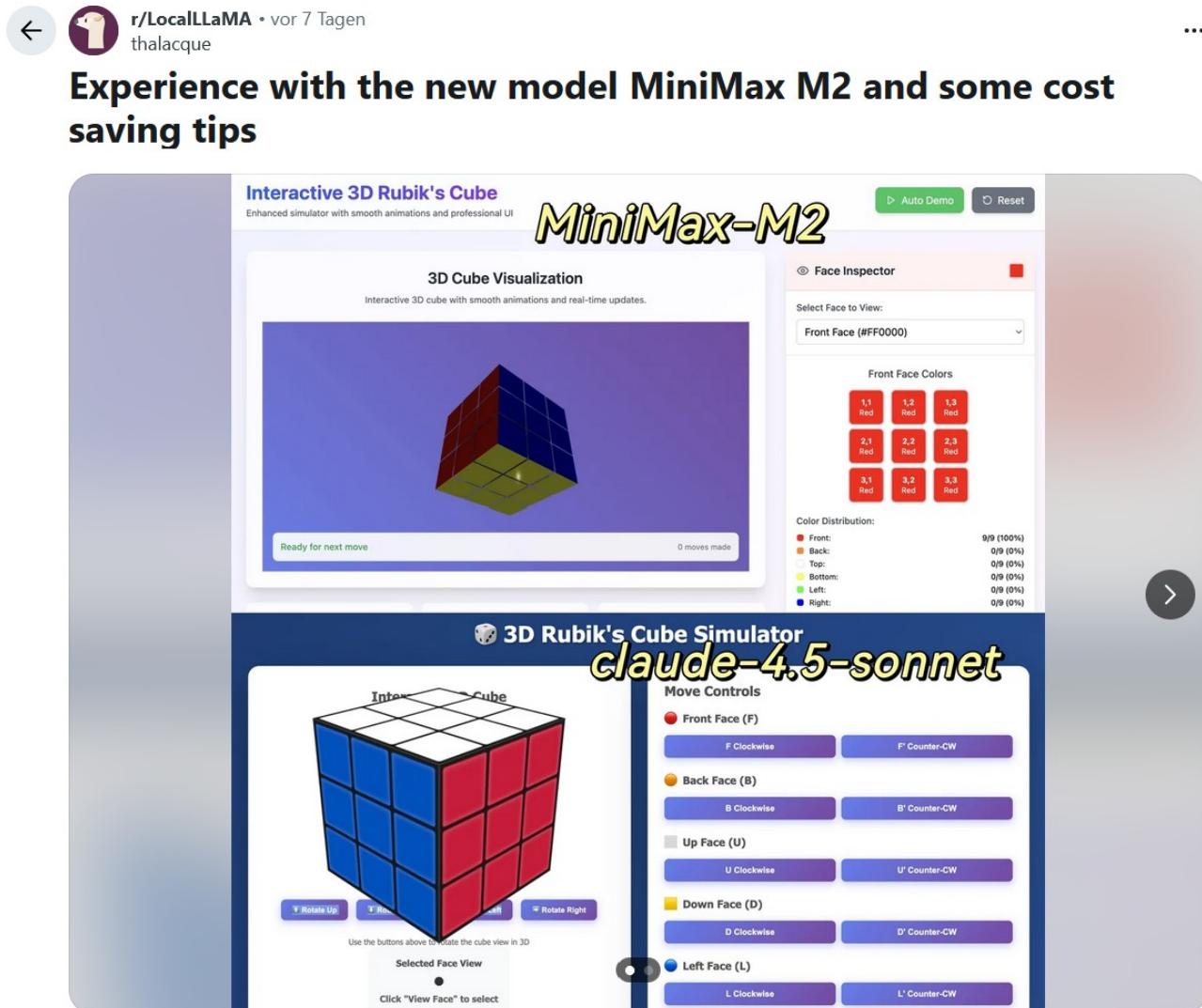
Output Speed vs. Price

Output Speed: Output Tokens per Second; Output Price: USD per 1M Tokens

Most attractive quadrant



„Aktuelles“ – They did it again: Modell MiniMax M2



„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

RESEARCH

Open source struggles to keep up



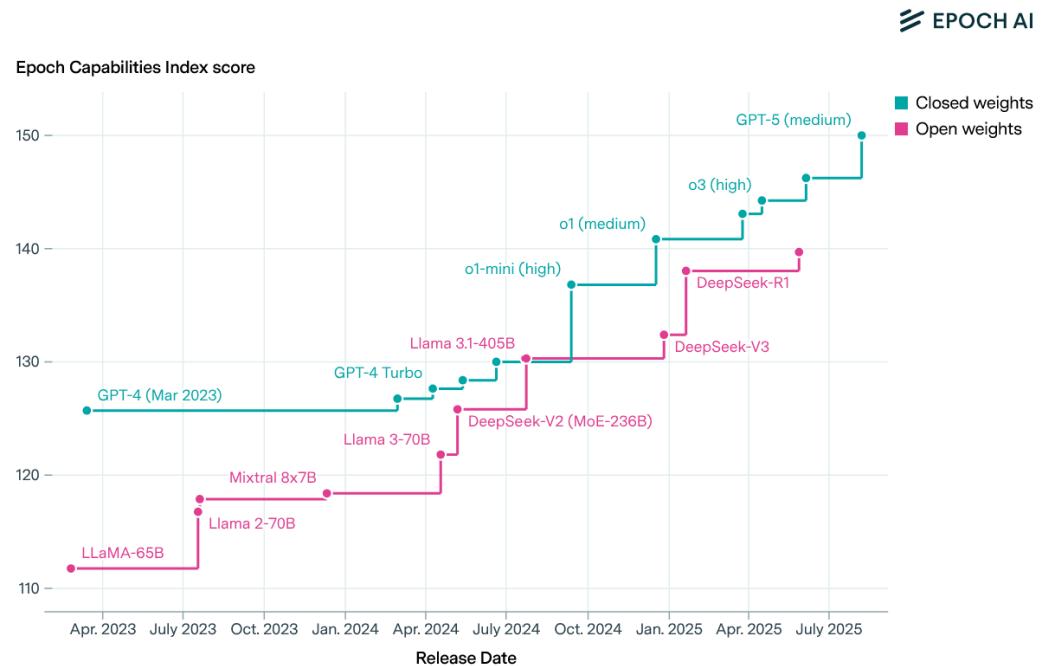
Open source models may be struggling to keep up with their closed-off counterparts.

According to data released Thursday by the [research institute Epoch AI](#), open weight models tend to lag around 3 months behind closed source models in capability.

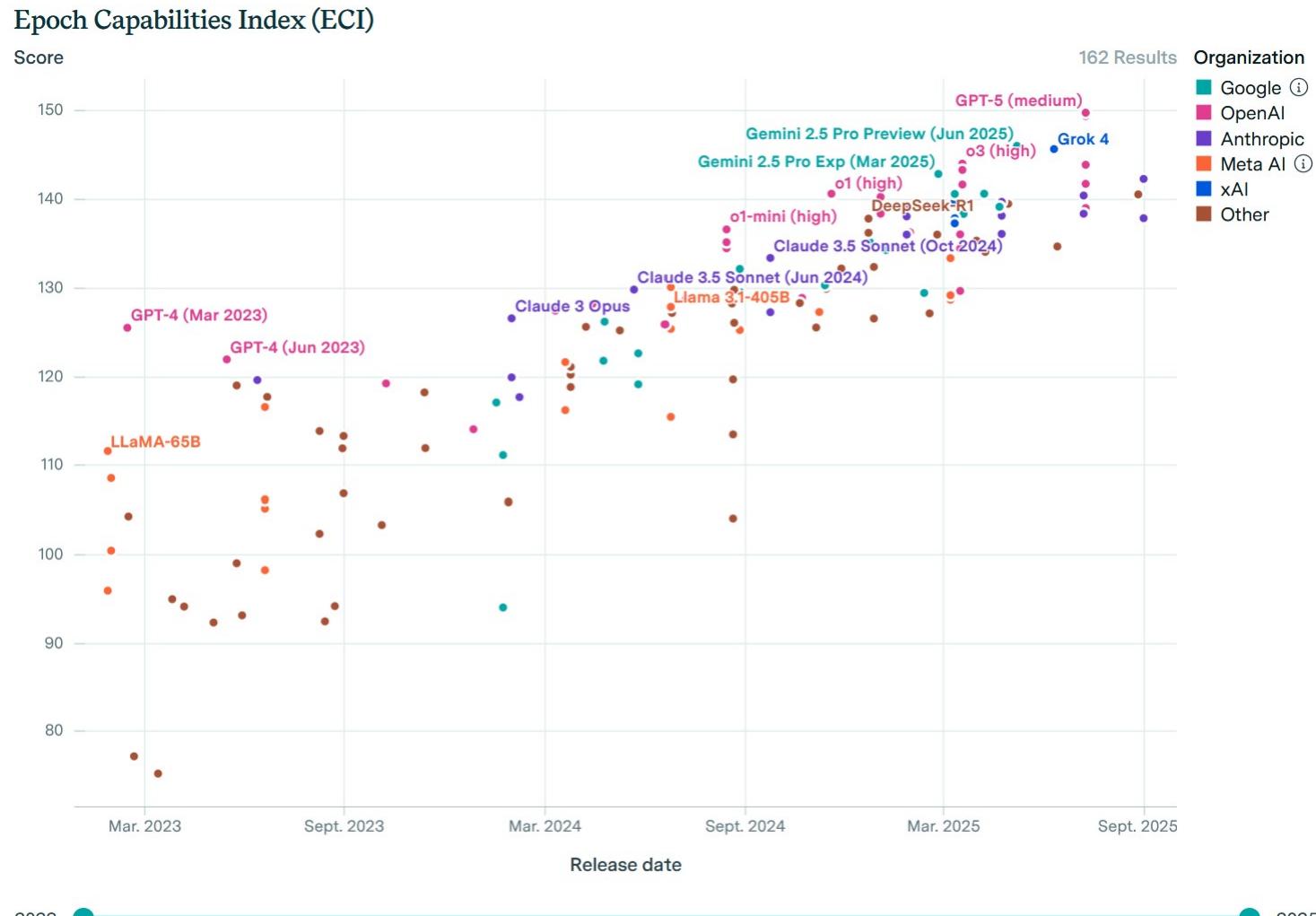
Using the institute's [Epoch Capabilities Index](#), which measures model capabilities across companies like Google, OpenAI, Anthropic, Meta and xAI, open-weight models score an average of seven points lower than closed source ones. Epoch notes that this is roughly the capability gap between OpenAI's o3, released in mid-April of this year, and GPT-5, released in early August.

Open-weight models lag state-of-the-art by around 3 months on average

Frontier open-weight models lag behind the most capable models by an average of 3 months in the [Epoch Capabilities Index \(ECI\)](#), our holistic measure of model capability. That corresponds to an average ECI gap of around 7 points, similar to the gap between o3 and GPT-5.



„Aktuelles“ – Open Source kommt immer mehr unter Druck ...



„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

Stanford University
Human-Centered
Artificial Intelligence

About ▾ Research ▾ Education ▾ Policy ▾ AI Index ▾ News Events Industry Centers & Labs 

NEWS

Universities Must Reclaim AI Research for the Public Good

DATE OCTOBER 30, 2025



With corporate AI labs turning inward, academia must carry forward the mantle of open science.

Ten years ago, Mark Zuckerberg made a surprise appearance at the academic conference NeurIPS, announcing the launch of Facebook's Fundamental AI Research unit (FAIR), signalling that AI research had leapt from university labs into the heart of Big Tech.

Fast-forward to today: Meta [announced drastic cuts](#) to FAIR even as AI becomes a trillion-dollar global industry. DeepMind no longer publishes technical details of their leading AI models and [has introduced six-month embargoes](#) and stricter internal review of papers to maintain competitive advantage. Similarly, OpenAI is now ClosedAI and, like other corporate labs, increasingly favor tech blogs and internal product rollouts rather than peer-reviewed publication or open-source release.

The tide of openness in AI is receding — and with it, the foundation of scientific progress itself.

SHARE    

AUTHORS

 John Etchemendy
 James Landay
 Fei-Fei Li
 Christopher Manning

„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

- Aufruf des ***Human-Centered AI Institute der Stanford University*** an die akademische Welt, das Ökosystem zu bewahren und weiter zu tragen, dass die KI-Entwicklung so möglich gemacht hat.
- Die privaten KI-Abteilungen (z. B. bei großen Tech-Firmen) ziehen sich zunehmend aus der offenen Wissenschaft zurück - weniger Veröffentlichung, weniger Open Source:
 - OpenAI → ClosedAI
 - DeepMind veröffentlicht keine Details mehr über ihre führenden Modelle und hat eine 6-monatiges Veröffentlichungsfenster, sowie strikte interne Reviews implementiert, um einen Wettbewerbsvorteil zu halten
- Offene Wissenschaft (Publikationen, Daten, Code, Modelle) war und ist eine zentrale Grundlage für Fortschritte in der KI-Forschung.

„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

How Openness Built Modern AI

The history of artificial intelligence is inseparable from the history of open science:

- The back-propagation algorithm, first shared openly in the 1980s, enabled deep learning's revival.
- Successful deep learning techniques were then pioneered at universities, particularly for speech and image recognition in Geoff Hinton's lab at the University of Toronto.
- Open datasets like TIMIT, TREC, MNIST, ImageNet, and Stanford Alpaca provided reproducible benchmarks and common ground for AI progress.
- Open-source code/libraries such as the Stanford CoreNLP toolkit and later TensorFlow, PyTorch, and FlashAttention offered free access to cutting-edge techniques.
- Shared benchmarks and challenges (e.g., GLUE, ImageNet competitions) trained generations of AI researchers and engineers.

- Wenn KI-Forschung zunehmend proprietärer wird und von einigen wenigen Firmen kontrolliert wird, leidet die Allgemeinheit - weniger Teilhabe, weniger Reproduzierbarkeit, weniger globale Kooperation.
- Universitäten haben eine besondere Rolle und Verantwortung. Sie sind nicht primär gewinnorientiert, sondern können Offenheit, Reproduzierbarkeit, Ausbildung und globale Teilhabe stärker betonen.

„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

- ***Probleme der aktuellen Entwicklung:***
 - Firmen-KI-Abteilungen stehen unter großem kommerziellem Druck durch die Konkurrenz, Kosten für die Modelle, ... ect.
 - Talente werden mit enorm hohen Vergütungen abgeworben – wodurch die Universitäten im Wettbewerb um Spitzenkräfte benachteiligt werden.
 - Universitäten haben meistens nicht genügend Rechenressourcen, Daten oder Infrastruktur, um mit den kommerziellen Entwicklungsabteilungen mithalten zu können – was die langfristig Ausbildung, die Forschung und das allgemeine Grundlagenwissen schwächt

„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

- **Aufruf an die akademische Welt:**
 - Universitäten müssen ihre Rolle für eine offenen KI-Forschung stärker übernehmen, um das öffentliche Interesse zu wahren.
 - Investitionen in offen zugängliche Daten, offene Modelle, frei zugängliche Ressourcen für Forschung und Lehre.
 - Aufbau globaler Partnerschaften – über Ländergrenzen, Disziplinen hinweg – um Daten, Rechenkapazität, Expertise zu teilen.
 - Förderung interdisziplinärer Forschung - neben rein technischen Fragen auch die Integration von Ethik, Design und sozialen Wissenschaften, damit KI dem Menschen und gesellschaftlichen Werten dient.

„Aktuelles“ – Open Source kommt immer mehr unter Druck ...

Home / Innovation / Artificial Intelligence

Why open source may not survive the rise of generative AI

Generative AI may be eroding the foundation of open source software. Provenance, licensing, and reciprocity are breaking down.



Written by David Gewirtz, Senior Contributing Editor

Oct. 24, 2025 at 6:23 a.m. PT

ZDNET's key takeaways

- Generative AI is erasing open source code provenance.
- FOSS reciprocity collapses when attribution and ownership disappear.
- The commons that built AI may not survive its success.

„Aktuelles“ – Die häufigsten Gründe für „Closed Source“ oder das Label „Vertraulich“



r/ProgrammerHumor • vor 6 Tagen

PassFlat2947

...

theOnlyReasonNotToShare

Meme



Not sharing your
source code out
of greed



Not sharing your
source code out
of shame

VCS-Antipattern: GitHub Security

Abstract:

Over 100,000 GitHub repos have leaked API or cryptographic keys

Thousands of new API or cryptographic keys leak via GitHub projects every day.



By Catalin Cimpanu for Zero Day | March 21, 2019 -- 23:21 GMT (23:21 GMT) | Topic: Security



A scan of billions of files from 13 percent of all GitHub public repositories over a period of six months has revealed that over 100,000 repos have leaked API tokens and cryptographic keys, with thousands of new repositories leaking new secrets on a daily basis.

MORE FROM CATALIN CIMPANU

Security
Norsk Hydro ransomware incident losses reach \$40 million after one week

Security
FTC asks broadband providers to disclose how they collect user data

Security
Top dark web marketplace will shut down next month

Security
ASUS releases fix for Live Update tool abused in ShadowHammer attack

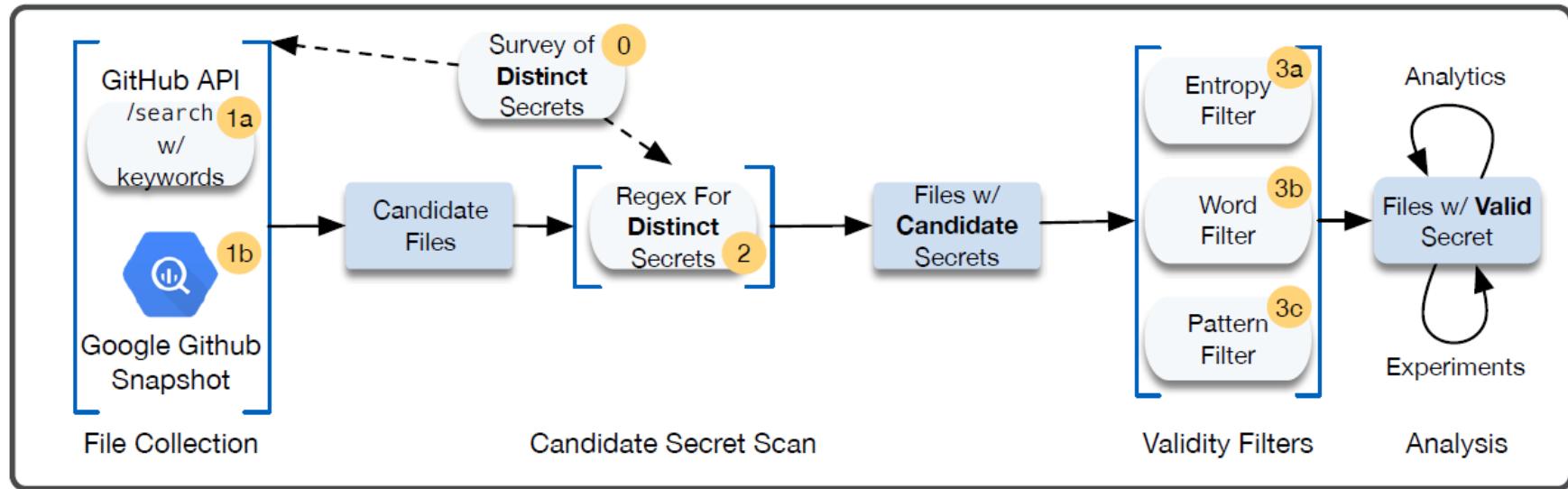
NEWSLETTERS



„We examine **billions of files** collected using two complementary approaches: a nearly **six-month scan of real-time public GitHub commits** and a **public snapshot covering 13% of open-source repositories**. We focus on **private key files** and **11 high-impact platforms with distinctive API key formats**. ... We find that not only is secret leakage pervasive — affecting over 100,000 repositories — but that thousands of new, unique secrets are leaked every day.“

Meli, Michael, Matthew R. McNiece, und Bradley Reaves. „How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories“. In Proceedings 2019 Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2019. <https://doi.org/10.14722/ndss.2019.23418>.

... GitHub Security



- 1a) GitHub Search API – 6 Monate Realtime Daten von Oktober 2017 bis April 2018
- 1b) BigQuery GitHub Snapshot File Collection – wöchentlicher Snapshot aller OpenSource lizenzierten Repositories, vom 4. April 2018
- 2) Scan nach potentiellen Token und Schlüsseln
- 3) Validierung der Kandidaten von Phase 2

... GitHub Security

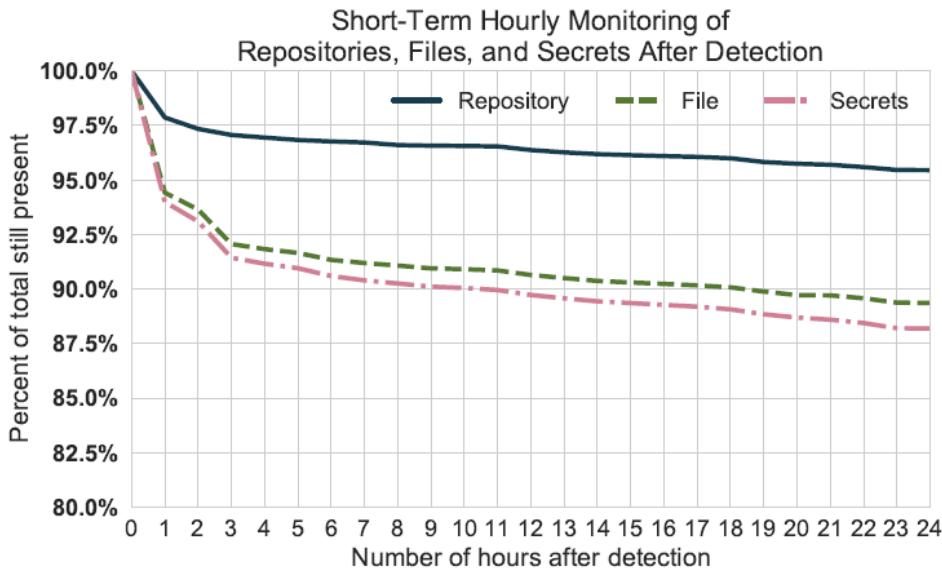
Domain	Platform/API	Key Type	Single-factor or Multi-factor	Primary Risks			
				Monetary Loss	Privacy	Data Integrity	Message Abuse
Social Media	Twitter	Access Token	M		X	X	X
	Facebook	Access Token	S		X	X	X
	YouTube ^a	API Key	S	X	X		
		OAuth ID	M		X	X	X
	Picatic	API Key	S		X	X	X
Finance	Stripe	Standard API Key	S	X		X	
		Restricted API Key	S				
	Square	Access Token	S	X		X	
		OAuth Secret	S				
	PayPal Braintree	Access Token	S	X		X	
Communications	Amazon MWS	Auth Token	M	X	X	X	X
	Gmail	(same as YouTube) ^a	(same as YouTube) ^a		X	X	X
	Twilio	API Key	S		X	X	X
	MailGun	API Key	S		X	X	X
	MailChimp	API Key	S		X	X	X
Storage	Google Drive	(same as YouTube) ^a	(same as YouTube) ^a		X	X	
IaaS	Amazon AWS	Access Key ID	S	X	X	X	
	Google Cloud Platform	(same as YouTube) ^a	(same as YouTube) ^a	X	X	X	
Private Keys	RSA	Cryptographic key	M	X	X	X	X
	EC	Cryptographic key	M	X	X	X	X
	PGP	Cryptographic key	M	X	X	X	X
	General	Cryptographic key	M	X	X	X	X

^a These secrets share the same format as part of the Google platform, but have different risks and are thus considered different

... GitHub Security

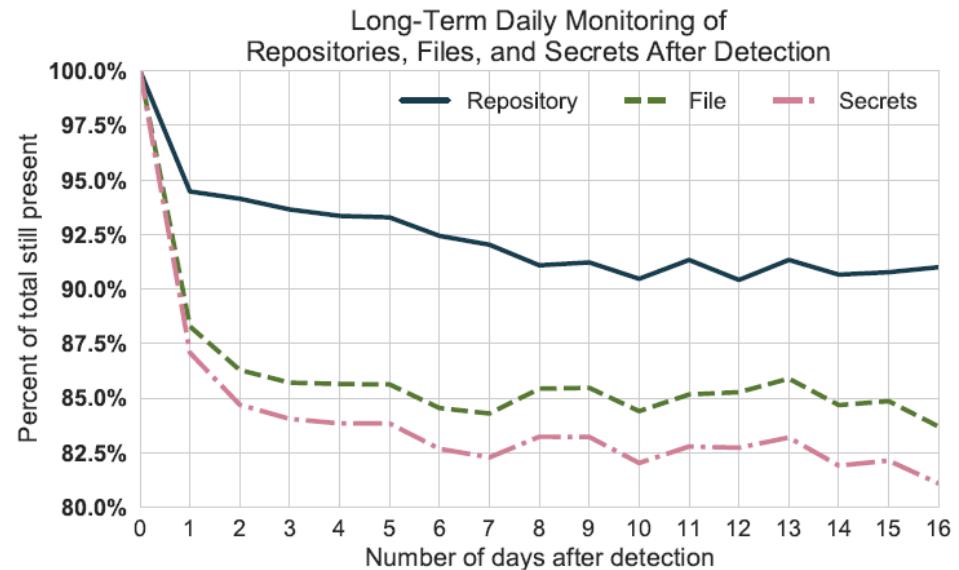
Secret	# Total	# Unique	% Single-Owner
Google API Key	212,892	85,311	95.10%
RSA Private Key	158,011	37,781	90.42%
Google OAuth ID	106,909	47,814	96.67%
General Private Key	30,286	12,576	88.99%
Amazon AWS Access Key ID	26,395	4,648	91.57%
Twitter Access Token	20,760	7,935	94.83%
EC Private Key	7,838	1,584	74.67%
Facebook Access Token	6,367	1,715	97.35%
PGP Private Key	2,091	684	82.58%
MailGun API Key	1,868	742	94.25%
MailChimp API Key	871	484	92.51%
Stripe Standard API Key	542	213	91.87%
Twilio API Key	320	50	90.00%
Square Access Token	121	61	96.67%
Square OAuth Secret	28	19	94.74%
Amazon MWS Auth Token	28	13	100.00%
Braintree Access Token	24	8	87.50%
Picatic API Key	5	4	100.00%
TOTAL	575,456	201,642	93.58%

... GitHub Security



(a) Many secrets are removed in the first few hours after being committed, but the majority remain

- Ca. 6% werden innerhalb der ersten Stunde „entfernt“
- Bleiben die fraglichen Stellen mehr als einen Tag, bleiben sie für längere Zeit
- Es werden kaum betroffene Repository's entfernt, sondern die betroffenen Files oder Stellen in Files – **noch in der History enthalten!!!**
- Es war keinerlei Rewrite-Versuch der History zu finden!!!



(b) Secrets that still exist on GitHub for a day after commit tend to stay on GitHub indefinitely

... GitHub Security

„Rewriting History Does Not Protect Secrets It is obvious that adversaries who monitor commits in real time can discover leaked secrets, even if they are naively removed. However, we discovered that even if commit histories are rewritten, secrets can still be recovered. In investigating the previous European case study, we discovered that we could recover the full contents of deleted commits from GitHub with only the commit's SHA-1 ID. Using our own repos, we experimentally confirmed **that this held true for both of GitHub's recommended methods for removing sensitive information**: *git filter-branch* or the *bfg* tool. The difficulty in this approach is in acquiring the commit hash, as it is hidden from GitHub's UI and Commits API. However, we found that these hidden commit hashes could be recovered with trivial effort via the Events API. Moreover, historical data from this API is available through the GHTorrent Project. Taken together, this indicates that the consequences of even rapidly detected secret disclosure is severe and difficult to mitigate short of deleting a repository or reissuing credentials.“

... GitHub Security

truffleHog

Searches through git repositories for secrets, digging deep into commit history and branches. This is effective at finding secrets accidentally committed.

NEW

truffleHog previously functioned by running entropy checks on git diffs. This functionality still exists, but high signal regex checks have been added, and the ability to suppress entropy checking has also been added.

These features help cut down on noise, and makes the tool easier to shove into a devops pipeline.

```
truffleHog --regex --entropy=False https://github.com/dxa4481/truffleHog.git
```

or

```
truffleHog file:///user/dxa4481/codeprojects/truffleHog/
```

```
Date: 2014-04-21 18:46:21
Branch: master
Commit: Removing aws keys

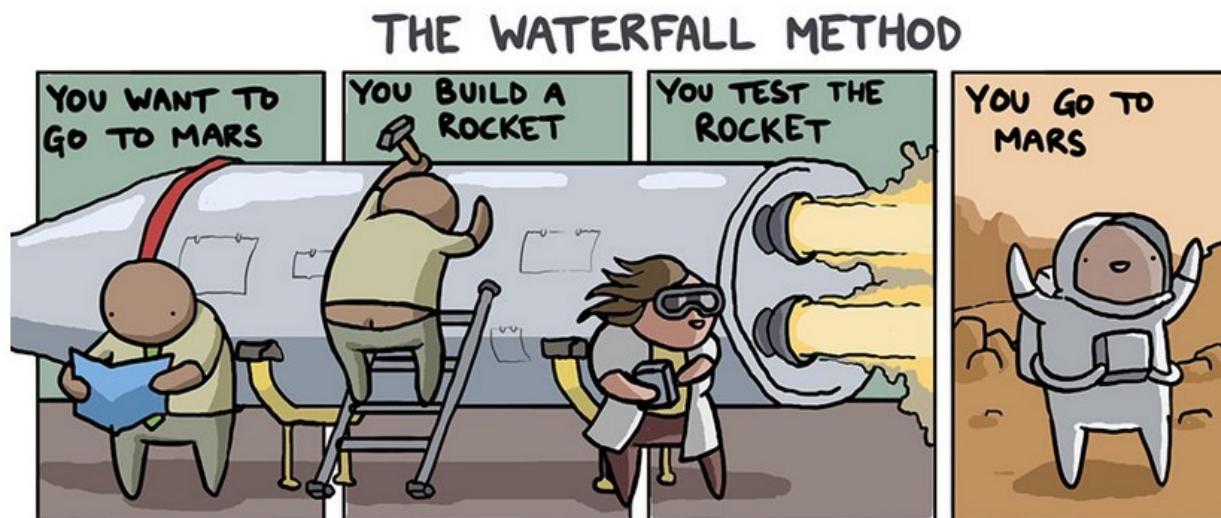
@@ -57,8 +57,8 @@ public class EurekaEVCacheTest extends AbstractEVCacheTest {
    //
    props.setProperty("awsRegion", "us-east-1");
    props.setProperty("awsAccessKeyId", "<aws access id>");
    props.setProperty("awsSecretKey", "<aws secret key>");
    props.setProperty("awsAccessKeyId", "AKIAJCK2WUHJ2653GNBQ");
    props.setProperty("awsSecretKey", "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");
    props.setProperty("appinfo.validateInstanceId", "false");

    props.setProperty("awsDiscoveryEndpoint", ".discovery.us-east-1.availabilityZones", "us-east-1c,us-east-1d,us-east-1e");
```

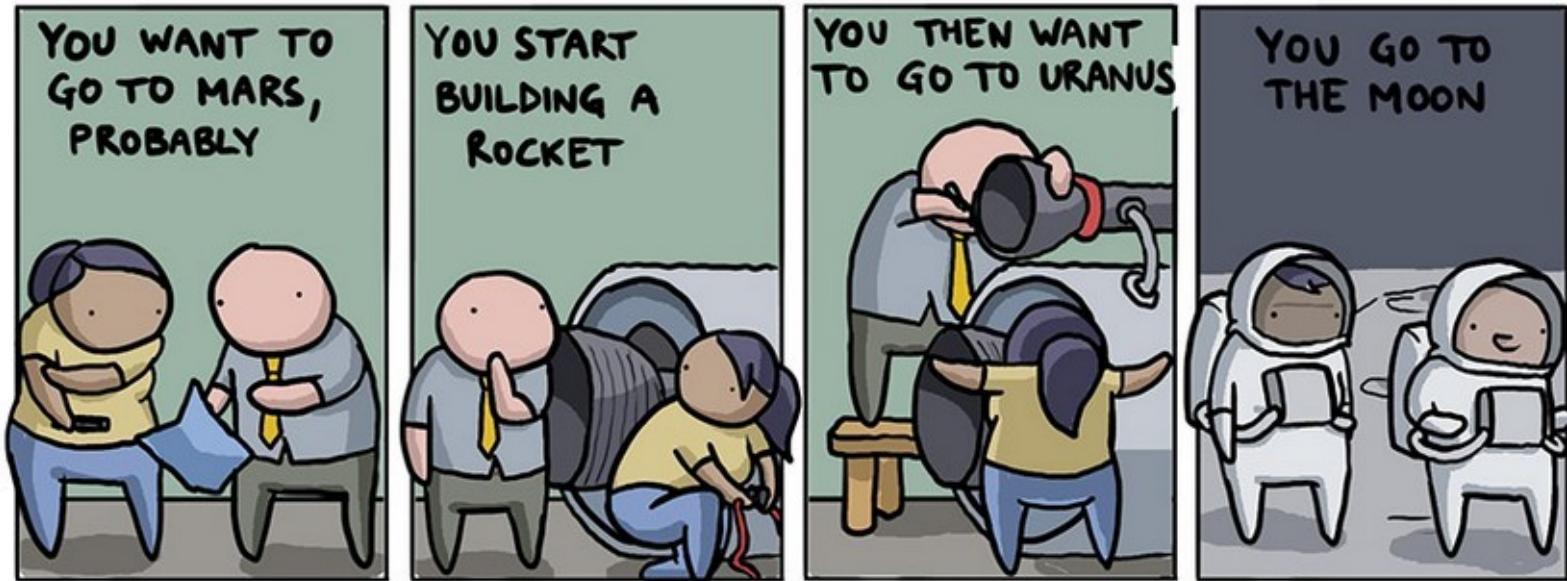
- TruffleHog prüft die Commits lokaler Repository's auf enthaltene Schlüssel oder Token über einen Entropie-Schwellenwert
- Erwies sich in diesem Paper aber nur als mäßig erfolgreich – „Our results show that TruffleHog is largely ineffective at detecting secrets, as its algorithm only detected 25.236% of the secrets in our Search dataset and 29.39% in the BigQuery dataset.“

Explaining Software Development Methods By Flying To Mars [comic]

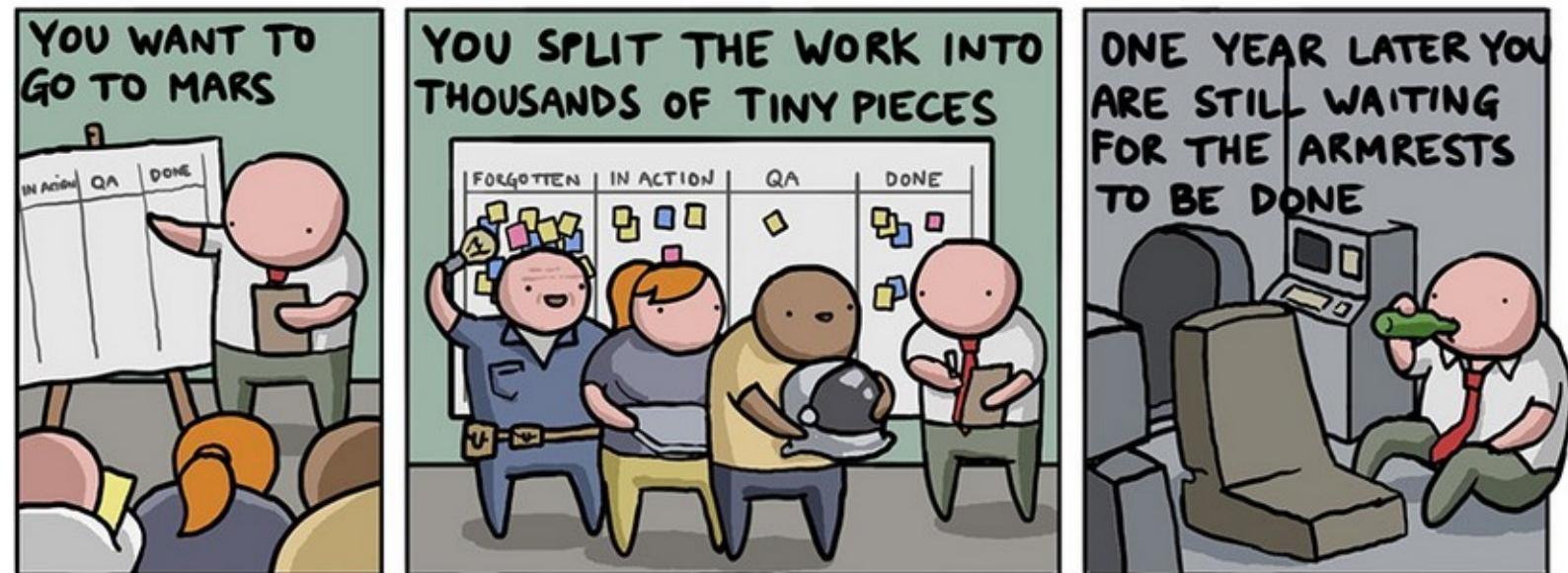
Elon Musk started out as a software engineer, and now he wants to go to Mars. Don't be on the first ship.



AGILE DEVELOPMENT



THE KANBAN METHOD



SCRUM



LEAN DEVELOPMENT

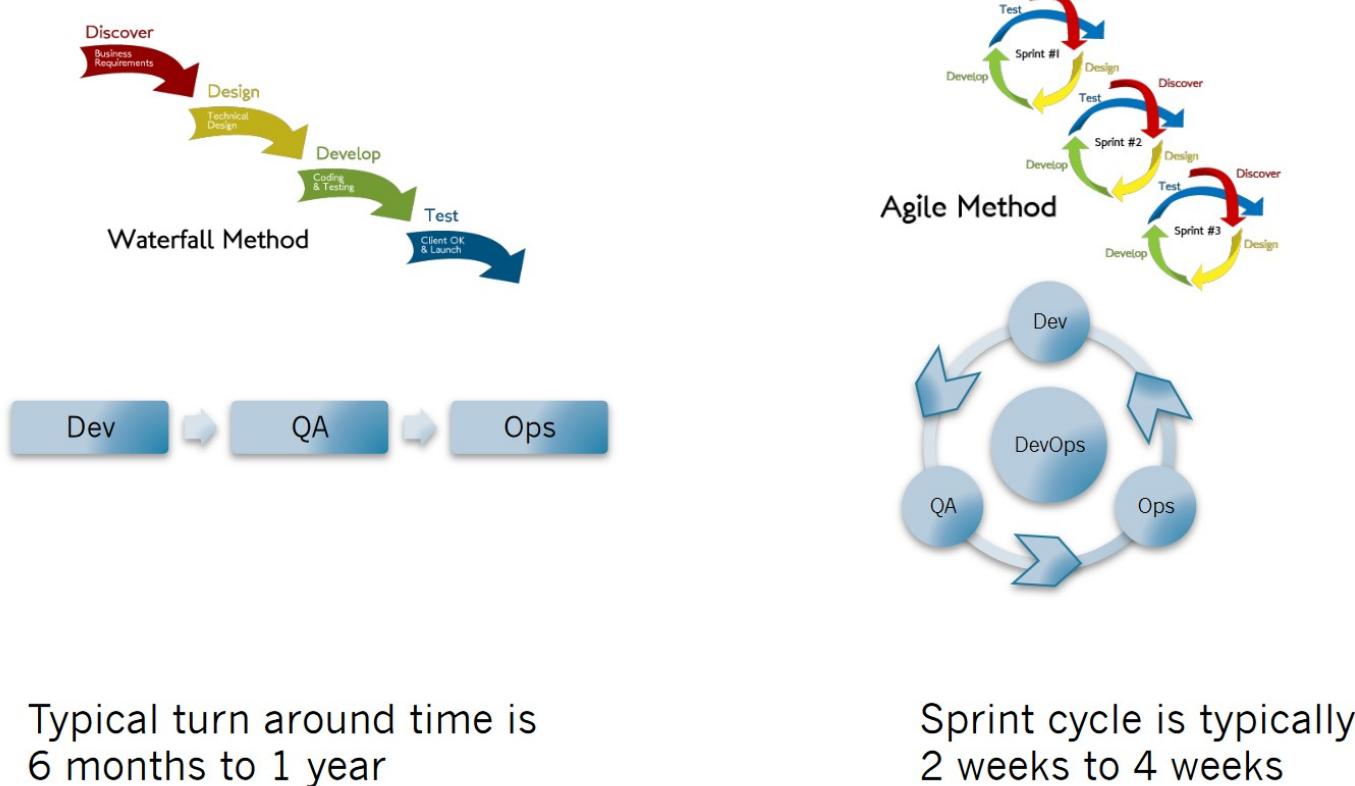


Truth be told, there's no wrong way to manage a big software project (except "moving fast and breaking things" – that one has not aged well). Waterfall may seem rigid and old school, but it's really just another way of saying "plan ahead". Agile methods are great for flexibility, as long as someone tells the client when to stop giving feedback.

4. Continuous Integration (**CI**) - Continuous Delivery (**CD**)

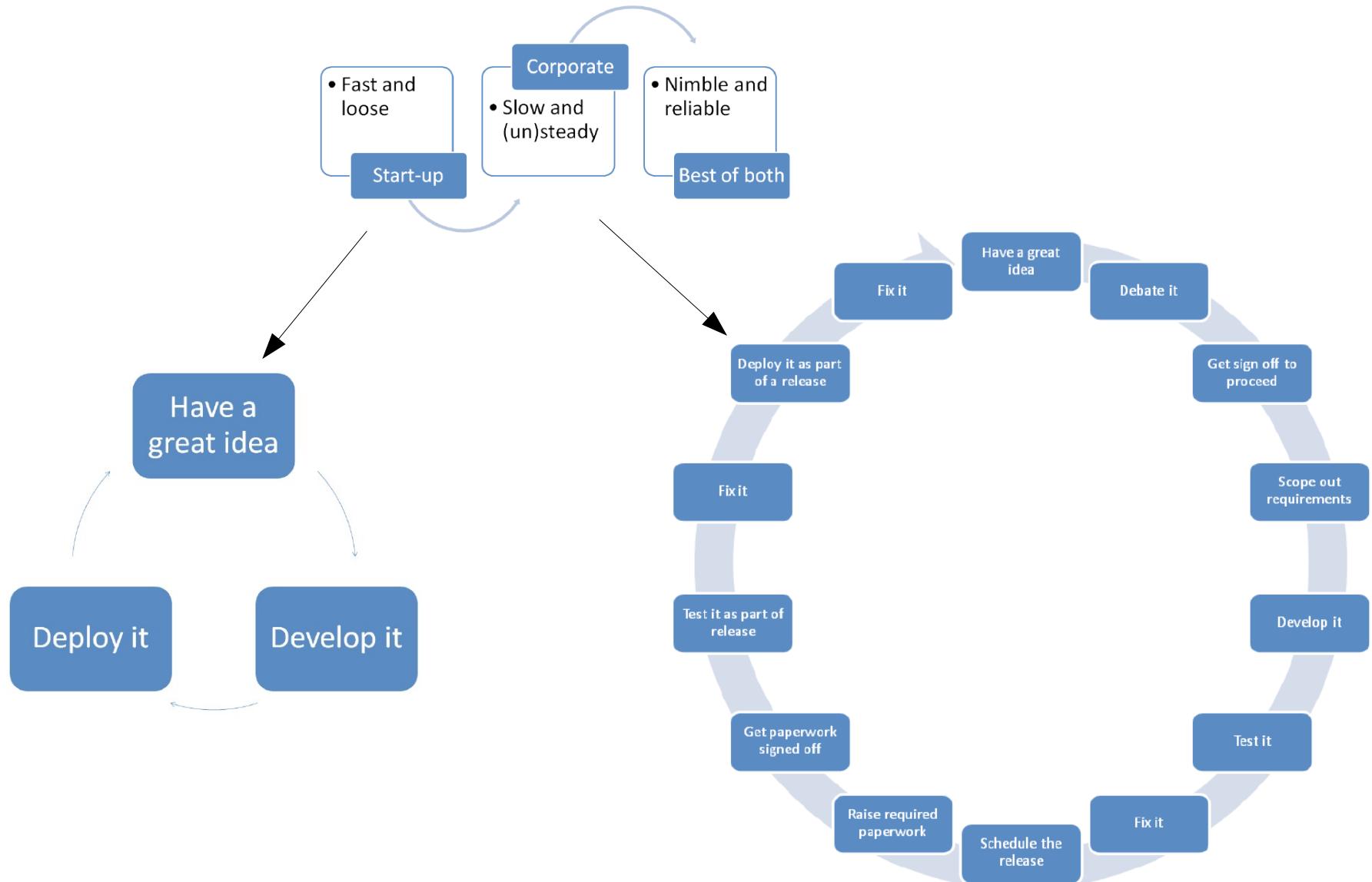
4.1. Vor betrachtungen

Zunehmender Übergang zu agilen Methoden



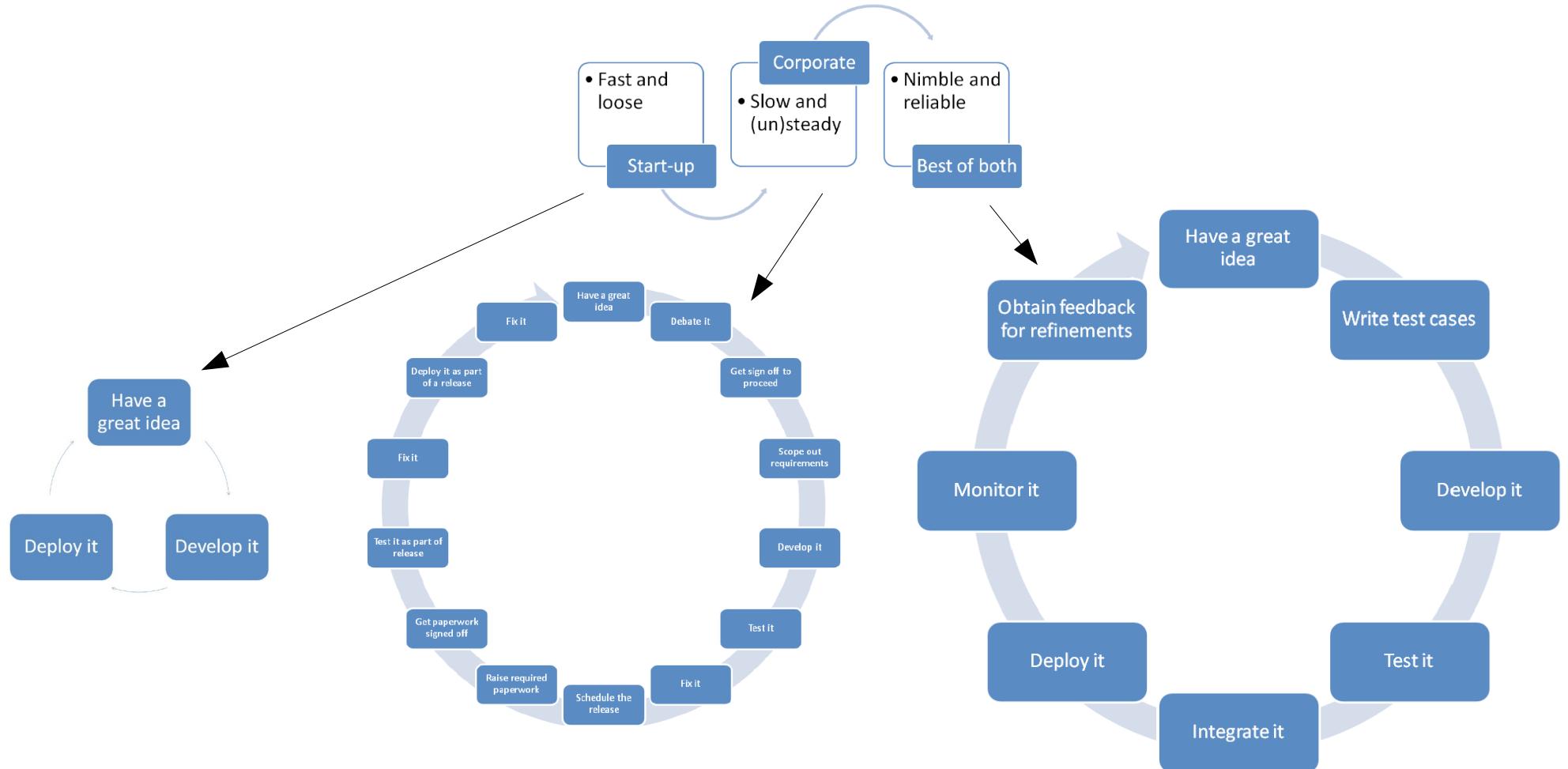
4.1. Vor betrachtungen

Entwicklungsprozess Start-Up vs. Großunternehmen



4.1. Vor betrachtungen

... Kompromiß

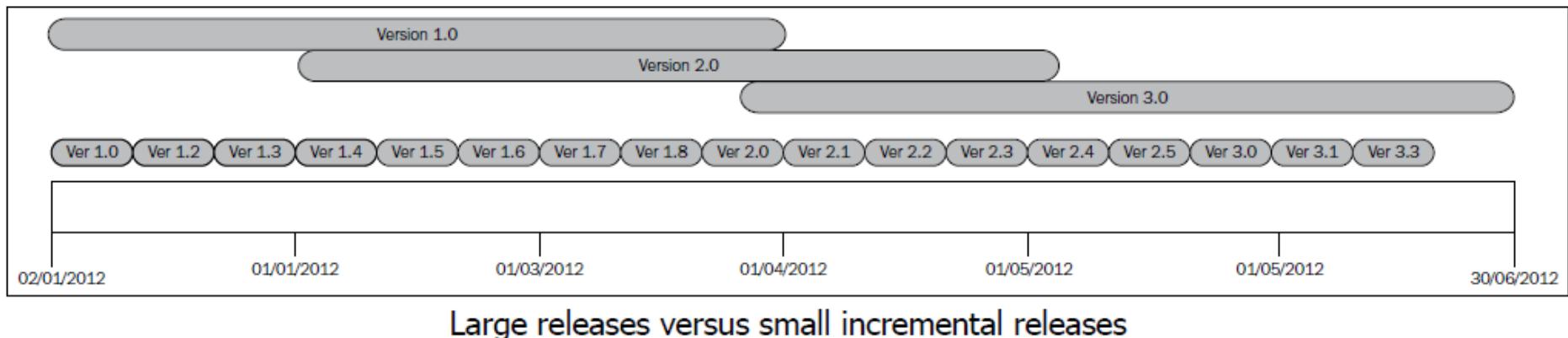


4.2. Was ist Continuous Integration (**CI**) - Continuous Delivery (**CD**)?

- Software-Engineering Ansatz (Sammlung von Prozessen, Techniken, Werkzeugen) mit dem ein Team in kurzen inkrementellen Zyklen Software entwickelt und dabei sicherstellt, dass zu jeder Zeit zuverlässig released werden kann. „Jeder Build ist ein potentielles Release!“
- Das bedeutet, das Bauen, Testen und Releasen von Software wird schneller und häufiger durchgeführt.
- Ziel ist ein klarer, wiederhol- und reproduzierbarer Prozess von der Änderung am Source-Code bis hin zum Deployment (Auslieferung) der fertigen Applikation.

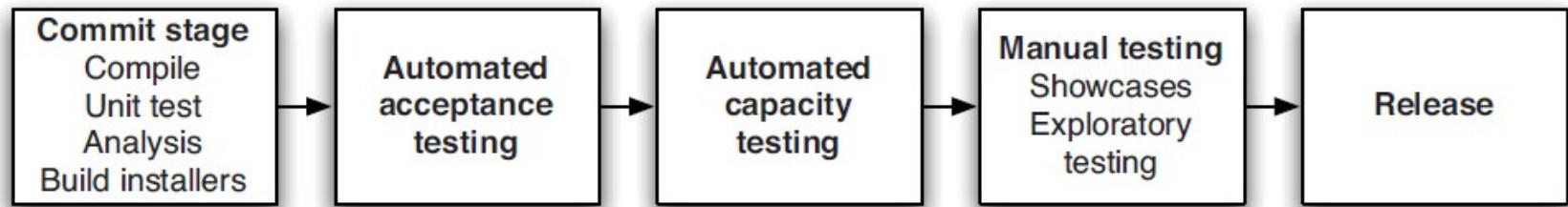
4.2. Was ist Continuous Integration (**CI**) - Continuous Delivery (**CD**)?

- Damit wird eine Senkung der Kosten, des Zeitaufwands und der Risiken bei der Auslieferung von Produkt-Updates erreicht (klein und kontinuierlich, statt „Big-Bang“):



- Zentraler Gegenstand ist die Deployment-Pipeline

4.2. Was ist Continuous Integration (**CI**) - Continuous Delivery (**CD**)?



- Deployment-Pipeline:
 - Jede Zustandsänderung (Source-Code, Konfiguration, Daten, Environment) führt zu einer neuen Instanz/Ausführung der Pipeline
 - In der ersten Stufe werden die Binaries und Installer erstellt, in den folgenden Teststufen wird sichergestellt, dass ein zuverlässiger Release-Kandidat erstellt wurde
 - Ist die logische Fortsetzung von Continuous Integration
→ CD setzt CI voraus

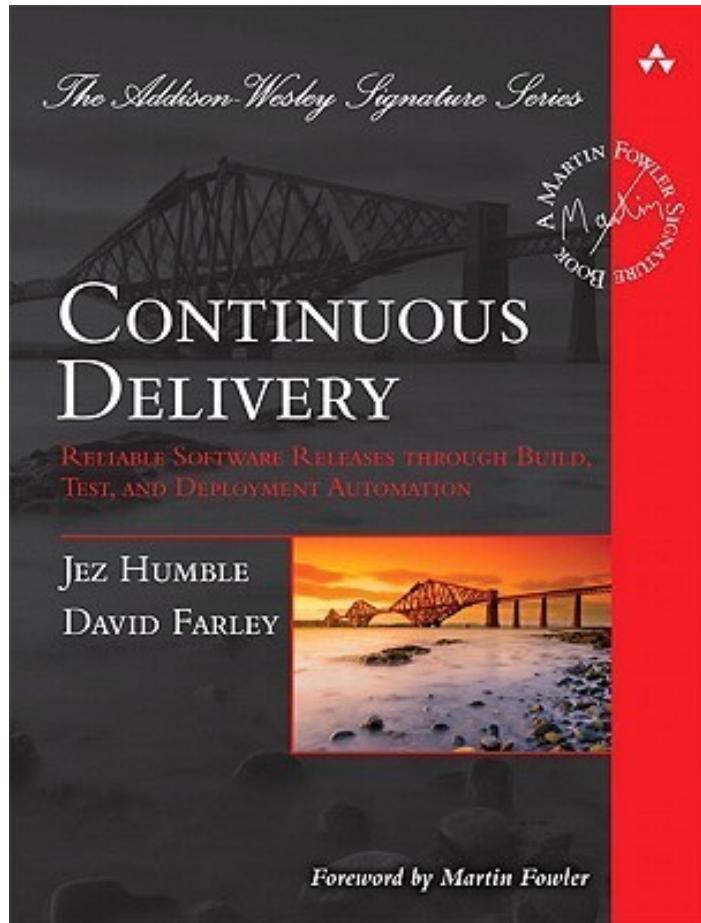
4.2. Was ist Continuous Integration (**CI**) - Continuous Delivery (**CD**)?

- Die Deployment-Pipeline sichert die folgenden drei wichtigen Punkte:
 - **Sichtbarkeit:** Alle Teile und Aspekte des Auslieferungssystems (Build, Tests, Deployment und Releasing) sind für alle Teammitglieder sichtbar
→ Verbesserung der Zusammenarbeit
 - **Schnelles Feedback:** Teammitglieder bekommen sehr zeitnah Rückmeldung über Fehlerzustände
 - **Kontinuierliches Ausliefern:** Durch eine umfangreiche Automatisierung kann ständig ein neues potentielles Release erstellt, getestet und deployt werden.

Noch einmal eine kurze begriffliche Abgrenzung:

- Continuous Delivery:
 - Ist eine Methode um komplett getestete, funktionsfähige Software in kleinen Inkrementen an die Produktionsplattform auszuliefern.
- Continuous Integration:
 - Ist eine Methode Software-Fehler so früh wie möglich im Entwicklungsprozess zu entdecken und sicherzustellen, dass alle Teile der der Applikation korrekt interagieren.
- DevOps:
 - Ist die Bestrebung Entwickler- und Operation-Teams möglichst eng kooperieren zu lassen, um den Softwareentwicklungs- und Auslieferungsprozess möglichst weit zu automatisieren.

„Das Buch“ zu Continuous Delivery



- 2011 von Jez Humble und David Farley veröffentlicht
- Machte den Begriff CD populär und „gab der Bewegung einen Namen“
- Gilt als grundlegendes Buch zu diesem Gebiet

4.3 Antipatterns im traditionellen Deployment

4.3.1 Manuelles Deployment

- Probleme:
 - Detaillierte Dokumentation erforderlich → zeitaufwendig und schwierig synchron zur Entwicklung des Deployment-Prozesses zu halten
 - Höhere Abhängigkeit von den Personen, die damit beschäftigt sind – „Single Point of Failure“
 - Release-Prozess dauert lang
 - Nicht dokumentierte AdHoc-Änderungen werden wahrscheinlicher
 - Deployment-Environments haben die Tendenz „auseinanderzulaufen“
 - Hoher Aufwand/Stress wenn schließlich ein Release gemacht werden soll bzw. muß

4.3.1 Manuelles Deployment

- Probleme:
 - Einerseit braucht man eine hohe Expertise, andererseits ist es ein sich monoton wiederholender Prozeß → förderte die menschliche Fehlerrate
 - ...
- Stattdessen:
 - Möglichst volle Automatisierung
 - Nur zwei Dinge sollten manuell gemacht werden, um eine Software in einer Ausführungsumgebung zu testen:
 - Auswahl der Version und der Ausführungsumgebung
 - „Deployment“ starten
 - Das Erstellen der Installer und Packages sollte ein vollautomatischer Prozeß sein.

4.3.2 Deployment in Produktions-ähnliche Umgebungen erst nach der SW-Entwicklung

- Probleme:
 - Wenn Tests bis hierher durchgeführt wurden, dann wurden sie höchstwahrscheinlich auf dem Entwickler- bzw. Entwicklungsrechner ausgeführt. → „Läuft auf meiner Maschine!“
 - Applikation kommt das erste Mal im Staging- oder Release-Bereich mit der produktiven Umgebung in Berührung → spätes Feedback bei Problemen
 - Geringe Interaktion zw. Operations-Spezialisten und Entwicklungsteam
 - Je länger der Release-Zyklus ist, desto länger kann das Entwicklungsteam von falschen Voraussetzungen ausgehen
 - Wird eine Applikation von vielen verschiedenen Teams (DB, Code, Konfiguration, ... etc.) entwickelt kommt hier die „große Integration“

4.3.2 Deployment in Produktions-ähnliche Umgebungen erst nach der SW-Entwicklung



- Probleme:
 - Je größer die Unterschiede zwischen Entwicklungsumgebung und Produktivumgebung sind, desto unrealistischer können die Annahmen des Entwicklungsteams sein → „Nur für das, was man sieht, kann man auch entwickeln!“
- Stattdessen:
 - Möglichst zu einem „Teil des Entwicklungsprozesses“ machen

4.3.3 Manuelle Konfiguration des produktiven Environments

- Probleme:
 - Deployment in den Staging-Bereich klappt, aber beim Produktivbereich funktioniert es plötzlich nicht
 - Unterschiedliche Versionsstände bei Laufzeitsystemen, Komponenten, Bibliotheken ... etc. zwischen den Test-, Staging- und Produktivumgebungen
 - Deployment kann damit auch eine Abhängigkeit von den handelnden (Operations-)Personen entwickeln
- Deshalb:
 - Alle Aspekte der Test-, Staging- und Produktivumgebungen, alle Konfigurationen am besten in der Versionskontrolle und durch einen automatisierten Prozeß angewendet → „Software as Infrastructure“ (Virtualisierung, Cloud, ...)
 - Konfigurationsmanagement → jeden Teil der von der Software genutzten Infrastruktur exakt reproduzieren

4.4 Zu implementierende Prinzipien

- ***Umfassende Automatisierung:***
 - Stellt ***Reproduzierbarkeit*** sicher
 - ***Alles Relevante in der Versionsverwaltung halten***
 - Geschwindigkeit
- ***Hohe Häufigkeit:***
 - Kleinere Inkremente → bessere Zuordnung von Fehlerursachen, schnelleres Feedback
 - „If it hurts, do it more frequently and bring the pain forward!“ → führt zur Gewohnheit und senkt den Stresspegel

4.4 Zu implementierende Prinzipien

- Jegliche Änderung an der Applikation muß zu Feedback führen:
 - Applikation besteht aus ausführbarem Code, Konfiguration, der „Laufzeitumgebung“ und Daten → Änderungen an diesen Bestandteilen → Änderung des Verhaltens der Applikation → Verifikation
 - z.B. Änderung des Source-Codes → Änderung des ausführbaren Codes → Bauen und Testen für jeden Check-In → entspricht Continuous Integration → Vorbedingung für CD
 - Und genau dieser erstellte Build wird dann im Rest der Pipeline getestet – kein spezieller Rebuild, jede potentielle Änderung muß kontrolliert werden

4.4 Zu implementierende Prinzipien

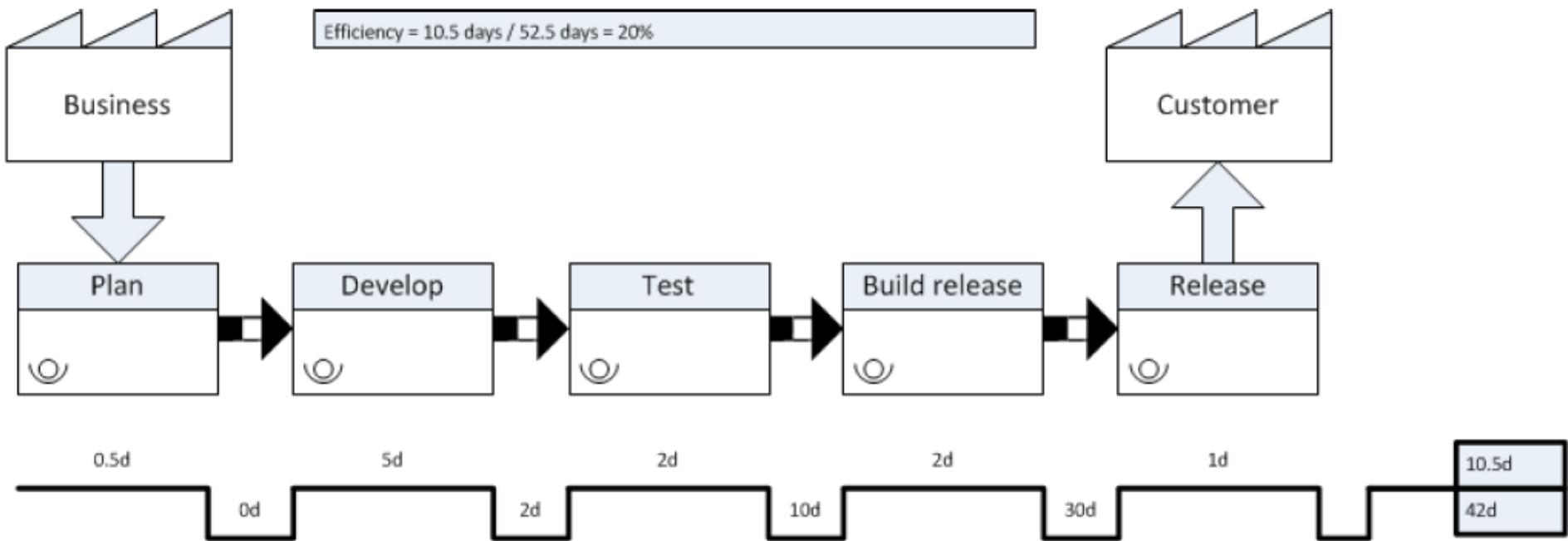
- Feedbackzyklus muß so kurz wie möglich sein
 - Durch die Automation nicht abhängig von der Anzahl vorhandener Mitarbeiter → „HW kann genutzt werden“
 - Eventuell parallelisieren
- Das ganze Team muß Verantwortung für den kompletten Prozeß übernehmen und zeitnah reagieren → zu etablierende Kultur
- Mit dem Ziel „Continuous Improvement“

4.5. Die Deployment-Pipeline

4.5.1 Value-Stream-Map (VSM)

- ***Value Stream Mapping*** kommt aus der Lean-Production, das vor allem durch die erfolgreiche Anwendung bei Toyota bekannt wurde – aber nicht von Toyota erfunden
- Vor allem auch übertragen auf „Knowledge-Work“-basierte Industriezweige, wie Software-Entwicklung, IT, Marketing, ... etc.
- Generelle Aufgabe:
 - Visualisierung des Flusses vom „Rohstoff“ (Idee) zum fertigen Produkt
 - Darstellung der Prozesse, die „Wert erzeugen“ und die, die keinen „Wert hinzufügen“ → Verbesserungsmöglichkeiten zeigen

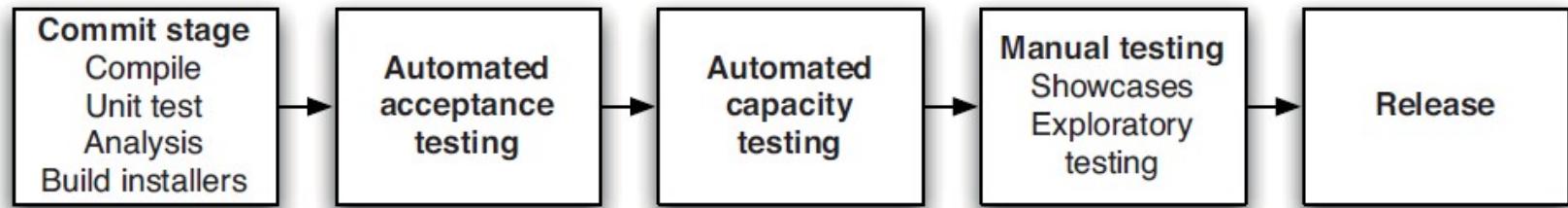
4.5.1 Value-Stream-Map (VSM)



This diagram also includes the efficiency (which is based upon the amount of time value is being added versus dead time within the flow)

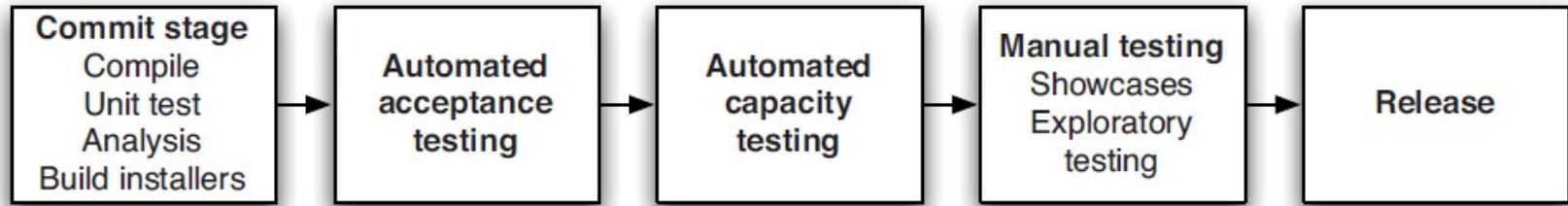
- Value Stream Map hier:
 - Von der Idee bis zur Auslieferung zum Kunden (Teil davon ist die Deployment-Pipeline)
 - Produktive Zeiten vs. „Standzeiten“
 - Visualisierung von „Silos“ und ihren Effekten

4.5.2 Deployment-Pipeline



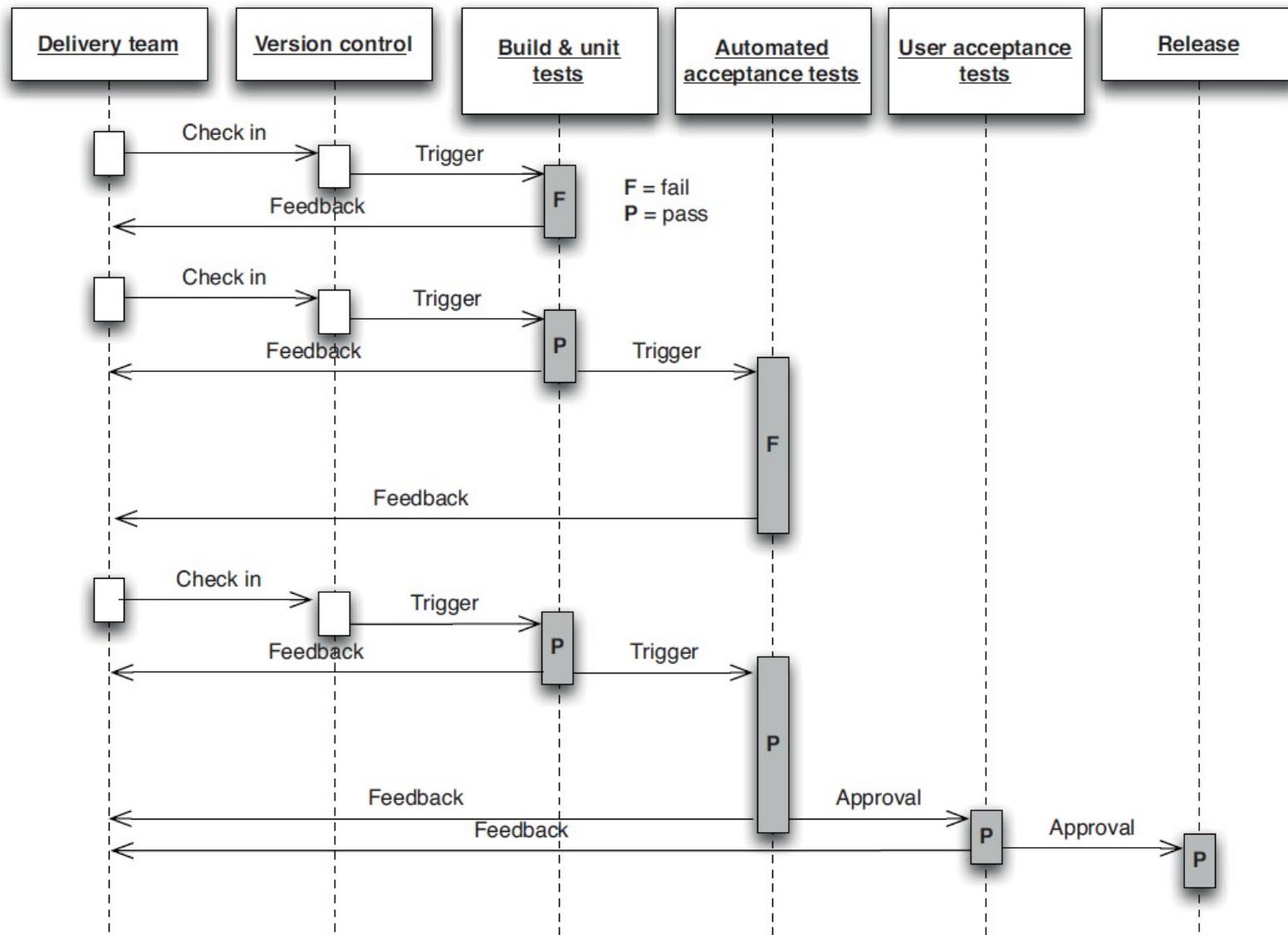
- 1. Stufe:
 - Bauen der Software → verifiziert die Syntax
 - Erste Tests (Unit-Tests) → verhält sich die Software prinzipiell wie erwartet
 - Bau der Installations- und Deploymentpackages
- 2. Stufe:
 - Funktionale Tests – liefert die Software auch Nutzen für die Anwendungsdomäne?
 - Stellt sicher, dass die Applikation auch aus Anwendungs- bzw. Anwendersicht seine Anforderungen erfüllt

4.5.2 Deployment-Pipeline

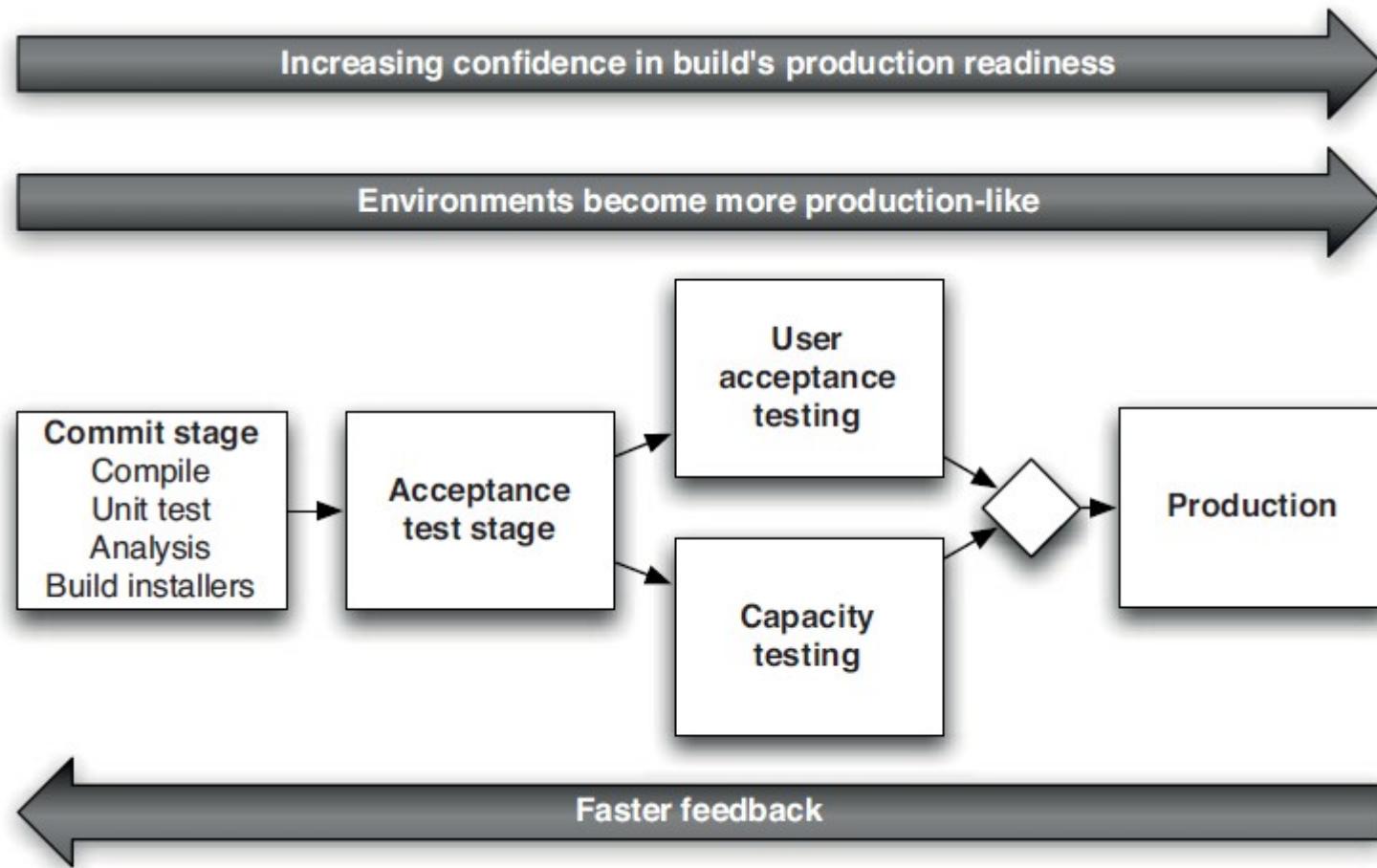


- **3. Stufe:**
 - Nichtfunktionale Tests – erfüllt die Applikation auch in Bezug auf die nichtfunktionalen Aspekte die Erwartungen ausreichend, z.B. in Bezug auf Ressourcenverbrauch, Geschwindigkeit, Verfügbarkeit, Robustheit, Sicherheit, ... etc.
- **4. Stufe:**
 - Manuelles, exploratives Testen durch ausgewählte Anwender
 - Identifikation fehlender Features oder Fehler (→ eventuell Konstruktion automatischer Tests für diese neu entdeckten Fehler und Integration in die Vorstufen)
 - User Acceptance Testing (UAT)

4.5.2 Deployment-Pipeline

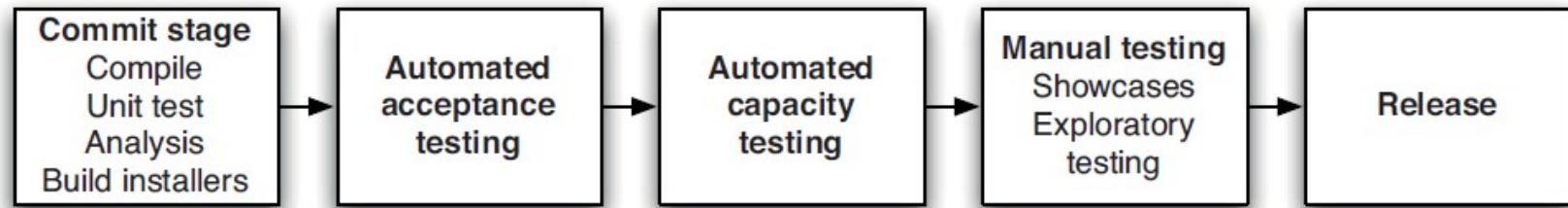


4.5.2 Deployment-Pipeline



- Kompromiß im Verlauf der Deployment-Pipeline zw. zunehmender Gewissheit und längerer Feedback-Zyklen

Noch einmal Begriffe ...



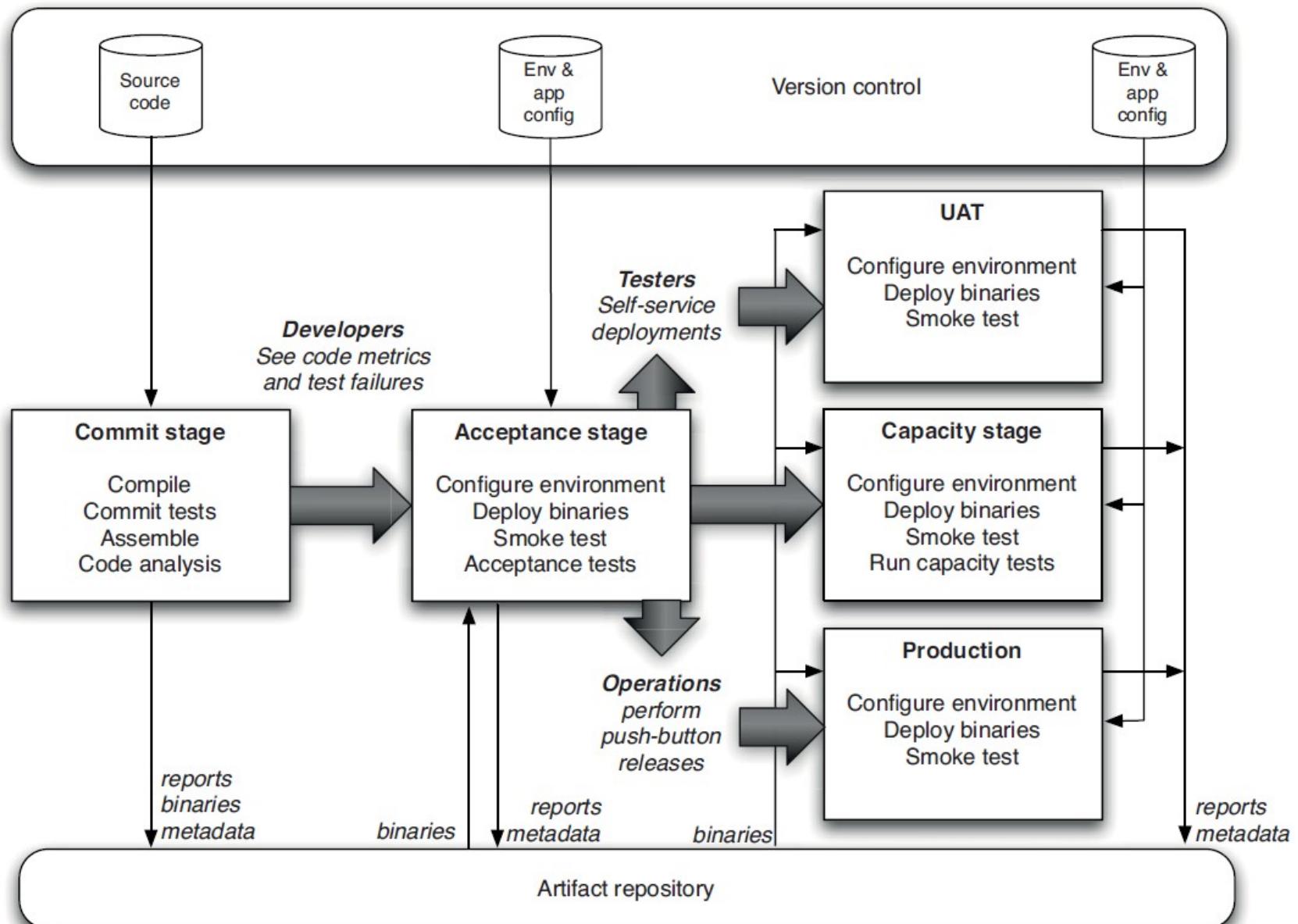
- ***Continuous Delivery:***

- Stellt die **Fähigkeit** dar, zu jeder Zeit in jede Umgebung der einzelnen Stufen zu deployen (inklusive Binaries, Änderungen der Konfigurationen, Änderungen der Konfiguration der Environments)
- D.h. nicht, dass ständig an Endkunden ausgeliefert wird
- Es geht um schnelles Feedback und Sichtbarkeit:
 - z.B. einen Bug-Report möglichst schnell nachvollziehen zu können (entsprechende Version und Konfiguration der Applikation, sowie zugehörigen Umgebung erzeugen und darauf ausführen)

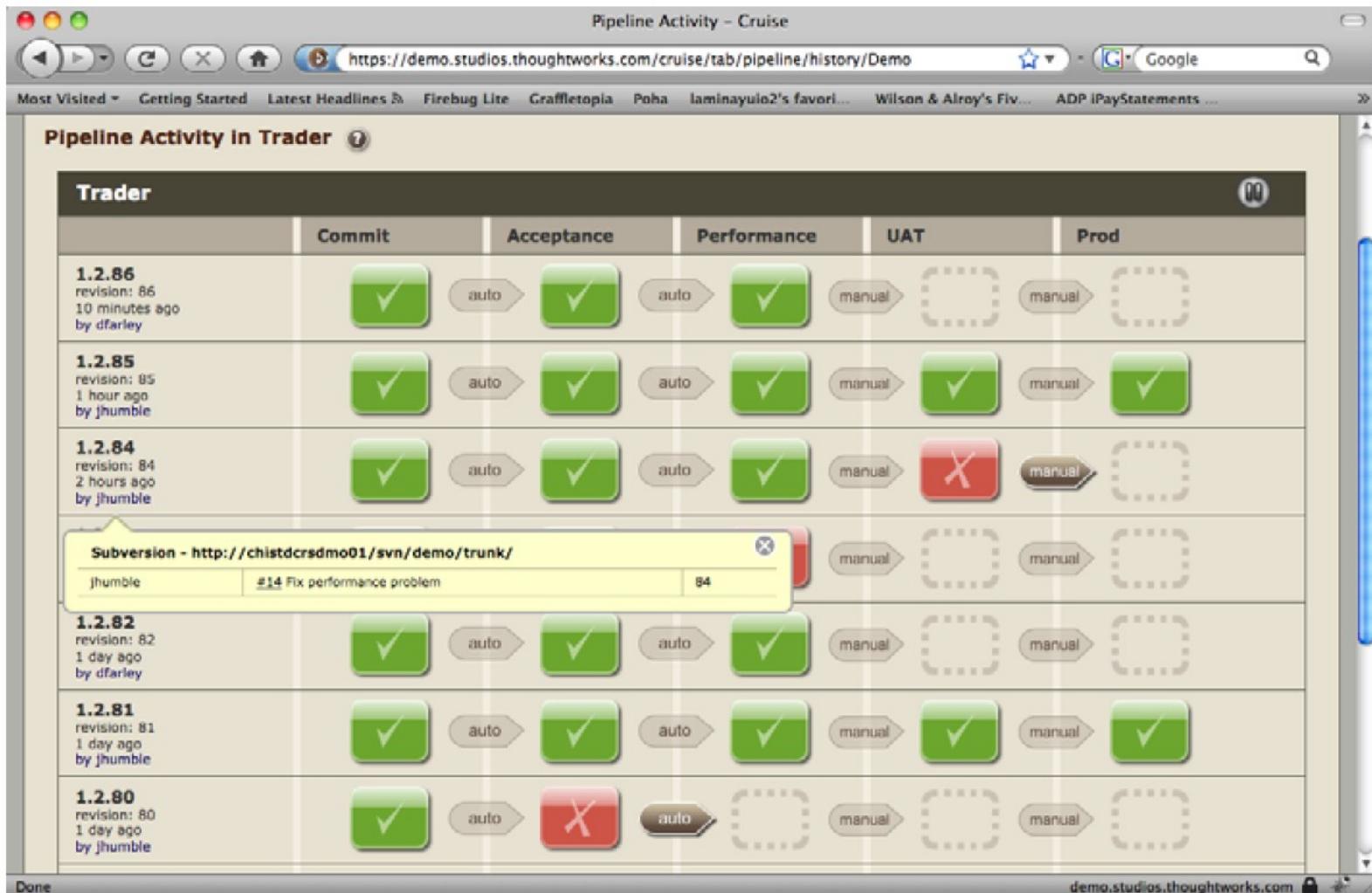
- ***Continuous Deployment:***

- Deployment tatsächlich bis zum Endkunden als wirkliches „Release“
- Für „Container-Applikationen“, Web-Applikationen oder Web-basierte API's (Microservices) möglich

4.5.2 Deployment-Pipeline



4.5.2 Deployment-Pipeline



Implementierung einer Deployment Pipeline

4.5.2 Deployment-Pipeline

Jenkins

Suc

Back to Dashboard

Status

Changes

Bauen mit Parametern

Pipeline löschen

Konfigurieren

Full Stage View

Open Blue Ocean

Rename

Pipeline Syntax

Pipeline DataManager

Application to convert data files generated by Multi Channel System software

Last Successful Artifacts

Multi Channel DataManager-1.10.5_x64.exe 91.94 MB anzeigen

Multi Channel DataManager-1.10.5_x86.exe 90.76 MB anzeigen

Recent Changes

Stage View

Declarative: Checkout SCM	Checkout	Build	Archive and copy installer to destination	Test Installers	Declarative: Post Actions
2s	986ms	2min 19s	4s	10min 44s	5s
10s	1s	2min 59s	4s	8min 58s	5s
1s	813ms	2min 27s	6s	7min 57s	4s
812ms	703ms	2min 7s	5s	8min 38s	5s
1s	750ms	2min 3s	4s	7min 27s	5s
1s	859ms	2min 10s	5s	11min 59s	4s

Build-Verlauf

Trend

suchen X

Average stage times:
(Average full run time: ~13min 26s)

1.10.5.55555-105 Mar 04 1 commit 13:41

1.10.5.18352-104 Dec 18 14 commits 13:45

1.10.5.18352-103 Dec 18 No Changes 11:57

1.10.4.18318-101 Dec 05 No Changes 10:19

1.10.3.18296-98 Oct 30 No Changes 09:56

1.10.3.18296-96 Dec 18 No Changes 15:09

1.10.2.18155-94 Dec 05 No Changes 11:25

1.10.1.18080-91 Dec 05 No Changes 10:19

1.10.0.18072-90 Oct 30 No Changes 11:37

1.9.8.17319-89 Oct 30 No Changes 09:56

Best Practices

- Binaries am Anfang bauen und durch die Pipeline schleusen:
 - Kein wiederholtes Generieren aus dem Source-Code, vermeidet das Einführen von Unterschieden – selbst unterschiedlich konfigurierter Compiler und Linker können Einfluß haben
 - In den späteren Stufen – keine anderen Versionen von binären Komponenten
 - Pipeline wird effizienter gehalten
- Keine umgebungsspezifischen Binaries
 - Gar nicht so selten
 - Hat komplexe Build-Systeme zur Folge
 - Restrukturieren → separieren einer umgebungspezifischen Konfiguration

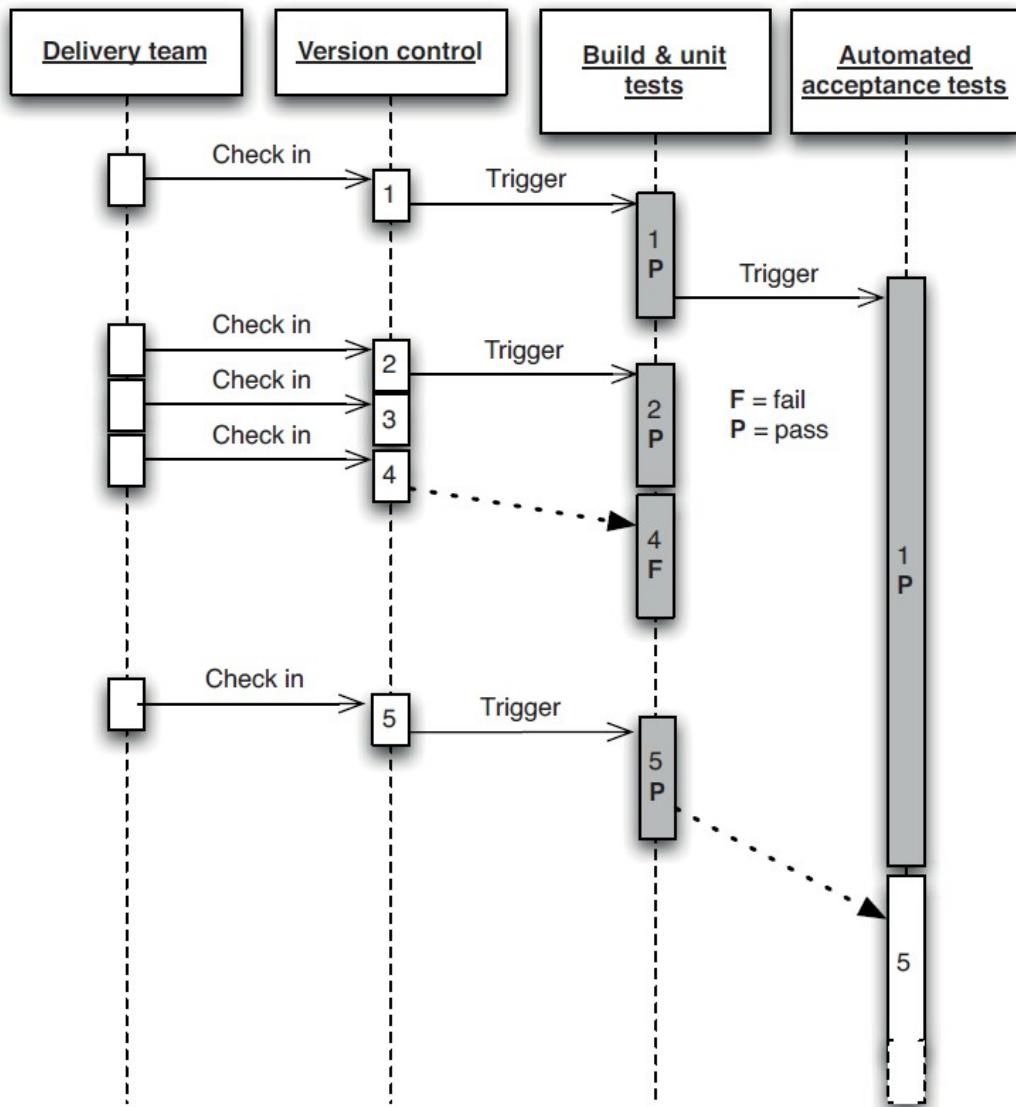
Best Practices

- Gleicher Deployment-Mechanismus für alle Umgebungen:
 - Sichtbare Trennung zw. den Dingen die gleich sind und denen, die sich ändern
 - Probleme können auf drei Ursachen zurückgeführt werden:
 - Settings im umgebungsspezifischen Konfigurationsfile der Anwendung
 - Probleme mit der Infrastruktur oder Services von denen die Applikation abhängt
 - Konfiguration der Umgebung
- Smoke-Tests der Deployments:
 - Smoke-Test ist ein erster grundlegender Test → startet die Applikation und läuft sie
 - Automatisches Testscript → prüft z.B. ob sich das Hauptfenster öffnet und der erwartete Inhalt da ist
 - Prüfen ob alle notwendigen Services da sind, z.B. Datenbank, Messaging-Bus, ... etc.

Best Practices

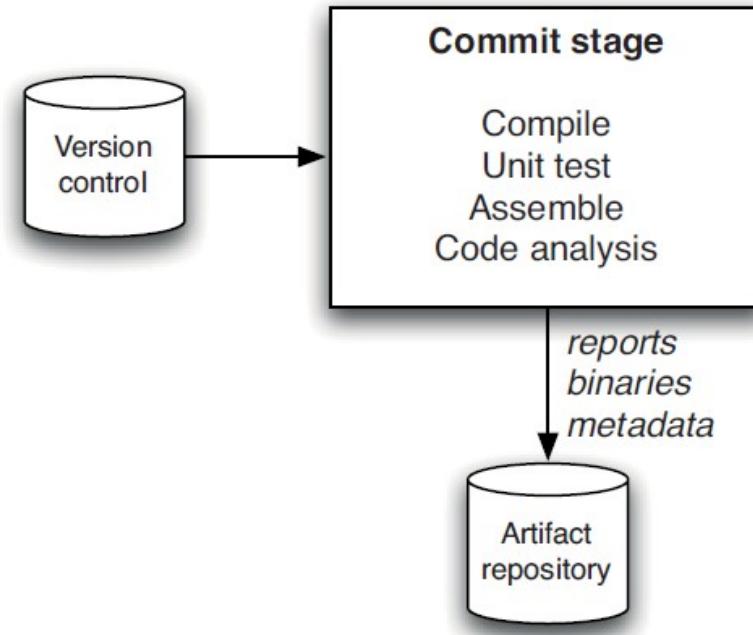
- In eine Kopie der Produktionsumgebung deployen:
 - Möglichst gleiche Infrastruktur, wie Netzwerktopologie, Firewall-Konfiguration, ...
 - Betriebssystem – gleiche Konfiguration, Version, Patchlevel, ... etc.
 - Application-Stack
 - Daten für die Anwendung

Best Practices



- Schneller Änderungsfluß durch die Deployment-Pipeline
- Der aktuellste Build/Artefakt wird weiter behandelt
- Tritt auf einer Stufe der Pipeline ein Fehler auf, so wird die ganze Pipeline gestoppt

4.6 Die Commit-Stage



- Die Commit-Stage beginnt mit einer Änderung im VCS und endet mit einem von zwei Ergebnissen:
 - Fehler-Report
 - Sammlung von ausführbaren Artefakten (Exe oder Dll), Reports, Metriken, Datenbank-Migrationen ...

4.6 Die Commit-Stage

- Ist meistens der Ausgangspunkt, wenn Continuous Delivery implementiert werden soll
- Ist Continuous Integration
- Sollte nicht allzu lange dauern – unter 5 bis 10 Minuten, um schnelles Feedback zu gewährleisten
- Überprüfen des Coding-Style
- Statische Code-Analyse / Metriken, z.B. Zyklomatische Komplexität (McCabe-Metrik), Duplikationen, Test-Coverage, ...

4.6 Die Commit Stage

4.6.1 VCS – Best Practices

- Regelmäßiges Einchecken in die Hauptlinie → regelmäßiges Integrieren, d.h. Feature-Banches so bald wie möglich wieder zurückführen
- Nicht in einen gebrochenen Build einchecken → diesen erst reparieren
- Die Unit-Tests lokal laufen lassen, die lokal überprüft werden können
- Keine Tests auskommentieren
- Aussagekräftige Commit-Messages

4.6 Die Commit Stage

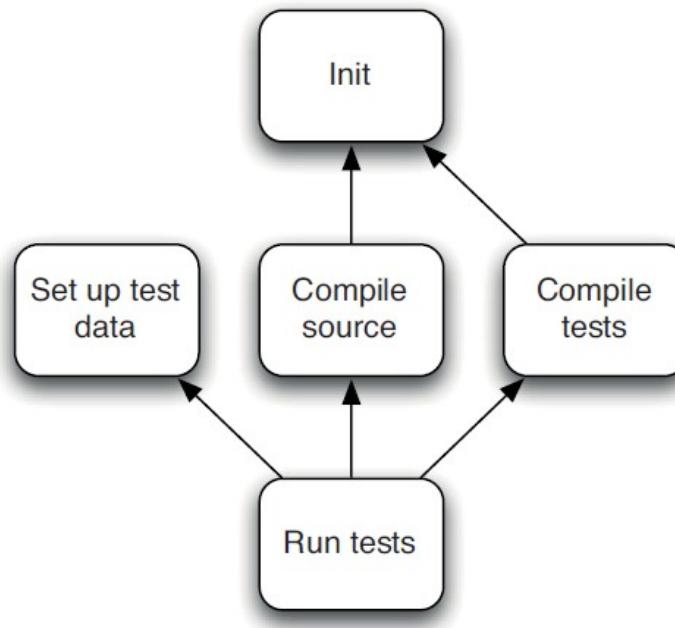
4.6.1 VCS – Best Practices

- Möglichst alles im bzw. in einem VCS, wie z.B. die Build-Scripte, Deployment-Scripte ...
- Konfigurationsscripte bzw. Konfiguration über separates VCS oder über Umgebungsvariablen → Orthogonaler Ansatz (ein Binary, viele Umgebungen)
- **Ganz wichtig!!!**: keine Credentials, Passwörter oder ähnliches im VCS oder hardcodiert im Source-Code

4.6.2 Automatisierter Build-Prozeß

- Kommandozeilen Interface (CLI) um die Software zu bauen, zu testen und zu deployen → Build-Scripte
- Build-Scripte gehören zur Code-Basis:
 - Werden getestet
 - Refactoring – vor allem wenn die Build-Prozesse komplexer werden
- Build-Scripte machen die Abhängigkeiten zw. den Komponenten transparenter und erleichtern so das Verständnis, das Debuggen und Pflegen der Applikation → erleichtert auch die Kooperation
- Nahezu jede Plattform hat ihr spezifisches Build-Tool bzw. Build-System, z.B. Make (C/C++), MSBuild (.NET), Rake (Ruby), Ant/Maven (Java), ...

4.6.2 Automatisierter Build-Prozeß



- **Build-Tools**
 - Modellieren ein Abhängigkeitsnetzwerk
 - Wird ein Target vorgegeben, wird ermittelt, welche Targets vorher in welcher Reihenfolge erreicht werden müssen

Build-Tools

- Task:
 - Das was direkt mit dem Task verbunden ist
 - Die Tasks von denen das Task abhängt
- Tasks werden zwar „abgelaufen“ aber nicht notwendigerweise ausgeführt falls sie noch „aktuell“ sind, z.B. „Init“-Task in der Abbildung
- Zwei Arten:
 - Task-orientiert, wie z.B. MSBuild, NAnt, Ant, ... - die Abhängigkeiten werden auf der Grundlage von Tasks formuliert
 - Produkt-orientiert, wie z.B. Make – die Abhängigkeiten werden auf der Grundlage von Produkten der einzelnen Tasks formuliert, wie Dll's oder Exe's
 - Timestamps von erzeugten Files → nur veränderte Files werden behandelt → ermöglicht inkrementelle Builds, was beispielsweise für C++ gut ist (komplexer Build-Vorgang, Templates, Optimierungen, ... etc. vs. Byte-Code/IL-Kompilate und anschließendem JIT-Compiler auf der VM)

Kurzer Einschub DSL

- Domänenspezifische Sprache (Domain-Specific Language/DSL):
 - Ist eine formale Sprache, die die Prozesse und Konzepte einer Anwendungsdomäne ausreichend spezifisch abbildet.
 - Gegensatz: universelle Programmiersprachen, wie z.B. C/C++, Java, C# ... etc.
 - Vorteile:
 - Von Domänen-Experten ohne großes Vorwissen bzw. Einschulungsaufwand zu bedienen
 - Deklarative Beschreibung möglich
 - Gute Lesbarkeit
 - Wenig Boilerplate-Code nötig
 - Einfachere Verifikation möglich

Kurzer Einschub DSL

- Domänenspezifische Sprache (Domain-Specific Language/DSL):
 - Nachteile:
 - Überhaupt eine neue Sprache → zusätzlicher Einarbeitungsaufwand
 - Risiko ob es genügend Unterstützung gibt bzw. geben kann – Community, Tools, „Ökosystem“ ...
 - „Locked-in“ falls es eine zu spezifische Sprache ist (Nische)
 - Evolution der Sprache – eventuell neue Konzepte nötig und damit zunehmend komplexer und unhandlicher
 - Passendes Abstraktionsniveau ...
 - Entwicklung einer Sprache samt Tools ist insgesamt nicht einfach → hohe Kompetenz und Erfahrung der Sprachentwickler nötig

Kurzer Einschub DSL

- Externe DSL's
 - Neu definierte Semantik und Syntax → ausdrucksstark und flexibel
 - Aufwändig in Implementation und Tool-Unterstützung → Language Workbench, wie z.B. Xtext (Eclipse) oder MPS (Meta Programming System) von JetBrains
 - Bsp.: SQL
- Interne oder eingebettete DSL's
 - Implementierung einer DSL innerhalb einer universellen Programmiersprache
 - Ist Untermenge der „Wirtssprache“
 - Damit ist der Implementierungsaufwand geringer und gleichzeitig bereits eine Tool-Unterstützung durch die Tools der „Wirtssprache“ gesichert (Editoren, IDE, ...)
 - z.B. DSL's in LISP bzw. insgesamt recht verbreitet in funktionalen Sprachen

Build-Tools

- Make (C/C++)
 - Mächtig, aber schnell unübersichtlich und komplex
 - Oft Top-Level Makefile, das rekursiv auf die Makefiles auf den Unterverzeichnissen aufruft
 - Zum Teil schwierige Fehler, z.B. statt führendem Tab Leerzeichen
- Ant (Java)
 - „Java-Antwort“ auf Make
 - XML-basiert → für Menschen nicht angenehm zu lesen
 - Ist plattformübergreifend
 - Externe DSL in XML
 - Zwar deklarativ im Wesentlichen, aber noch imperative Tags → kann in kompliziert zu verstehenden Mixturen resultieren
 - Kann recht lang werden

Build-Tools

- MSBuild
 - .NET-Antwort auf Ant (gibt auch NAnt)
 - Ebenfalls XML-basiert → ähnliche Probleme wie Ant
 - MSBuild hauptsächliches Build-Pattform auf dem MS-Stack
 - neben interessanten aber noch nicht so weit verbreiteten Alternativen wie z.B. FAKE (F#) oder Cake (C#)
 - Visual Studio benutzt MSBuild, aber MSBuild hängt nicht vom Visual Studio ab → läuft auch auf Linux und MacOS
 - Offene Entwicklung auf <https://github.com/Microsoft/msbuild>
- Maven (Java)
 - „Konvention vor Konfiguration“
 - Ebenfalls XML-basiert (Externe DSL in XML)
 - ...

Build-Tools

- Build-Tools auf der Basis von internen DSL's:
 - Alle Möglichkeiten einer allgemeinen Programmiersprache zugreifbar
 - Kein „Bruch“ zwischen Applikationsentwicklung und Build-Tool
 - Gute Unterstützung durch bereits vorhandene Tools
 - Refactoring und Modularisierung gut möglich
 - Bsp:
 - Rake (Ruby)
 - Psake (PowerShell-basiert)
 - Fake (F#)
 - Cake (C#)
 - Buildr
 - Gradle
 - ...

Build-Prozeß

- Immer relative Pfade nutzen
- Manuelle Schritte entfernen
- Nachvollziehbarkeit von Versionskontrolle zum Binary sicherstellen
 - z.B. durch Versionsmetadaten in Assembly's oder Jar's
 - Falls keine Unterstützung von vornherein → Hashes (z.B. MD 5) der Binaries zusammen mit Revisionsinformation speichern
- Keine Binaries in das VCS des Source-Codes
 - Generell keine „abgeleiteten Artefakte“ in das Source-Code VCS
 - Stattdessen ins *Artifact-Repository*