MISSION READY SOFTWARE

Predicting software reliability in an CI/CD environment

HTTP://WWW.MISSIONREADYSOFTWARE.COM

SALES@MISSIONREADYSOFTWARE.COM

321-514-4659

About Ann Marie Neufelder

- Authored the industry guidance on software FMEA - "Effective Application of Software Failure Modes Effects Analysis", published for CSIAC, 2014.
- Chairperson of IEEE 1633 Recommended Practices for Software Reliability Working Group (2016 edition) –See video for more information: https://www.youtube.com/watch?v=vmW2EM5KkMo&t=18s
- IEEE Lifetime Achievement Award, 2017, Reliability Society.
- 39 years of software engineering and software reliability experience
- Developed electronic warfare systems prior to starting Mission Ready Software
- Managed small and large software development and test teams throughout career and has applied virtually every development practice over 39 years
- Authored NASA's Software FMEA and software FTA training webinar
- Authored Intel's process for evaluating vendors with regards to software
- Co-authored USAF Rome Laboratory "System and Software Reliability Assurance Notebook", with Boeing Corp.
- Authored "Ensuring Software Reliability", Marcel-Dekker, 1993.
- Benchmarked 200+ software intensive systems for reliability, on time delivery and customer satisfaction. See video for more information. https://www.youtube.com/watch?v=HApDHxtG_Mk&t=1s
- Has analyzed almost 1 million failures due to software and categorized by failure mode and root cause. See video for more information. https://www.youtube.com/watch?v=XdrzT8b8qXs&t=20s
- U.S. Patent 5,374,731 for predictive model
- 1983 Graduate of Georgia Tech

About Mission Ready Software

- In business since 1991
- Customers include DoD, medical devices, energy, space, commercial aircraft, semiconductor, vehicles, major electronics
- Worlds largest software reliability benchmarking data
  - 679 factors measured at 200+ engineering organizations
- Worlds largest database of software failure events analyzed for root cause
  - The Department of Defense recently published our "Common Defect Enumeration" on DAU R&M CoP website
- Frestimate software developed in 1993 based on regression models for predicting software defects. Retired in 2021 and replaced by Requs AI Predict.
- Requs AI Predict is first machine learning tool to predict software defects before the code is written
- Requs AI Predict SFMEA is the first machine learning tool to auto generate a software FMEA

MISSION READY SOFTWARE
SOFTREL LLC

Agenda

- Software defects, failures, rework and probability of on time delivery can be predicted before the code is even written
- Practical applications for predictions
- How to use the models in DevSecOps

Models that predict defects, failures, rework, on time delivery

- If you have enough data - you can predict anything
  - 30 years ago weather forecasts for any given day had only 50% accuracy
  - Now you can accurately predict when to schedule your tee time
- A good predictive model has the following
  - As much data as possible from as many engineering companies as possible
    - How many defects were actually deployed and how long did it take for customer to discover them?
    - How did the organization develop the software?
    - What are the product characteristics? Inherent industry risks?
    - What was the experience level of the software organization?
    - What was the level of rigor of the testing?
    - Was the design and specifications clear and unambiguous?
- The more parameters the model has the more accurate it will be.
  - No one gets approved for a home mortgage based on only 1 factor.
  - You don't predict defects based on only 1 factor either.

History of predictive models

Model	# Data sets	# Inputs	Year last updated	Comments
USAF Rome Laboratory TR-92-52	53	Up to 212	1992	• Very outdated. 31 years in software engineering is akin to a millennium. • Difficult to use for non-aircraft systems
RIAC 217 Keene model	14	1	1998	• Model violated basic rules of statistical modeling • Model assumed that only 1 parameter is needed to predict defects • Not enough parameters or data sets
Mission Ready Software – Requs AI Predict (formerly Frestimate)	> 150	679	Continuously updated since 1993	• Mission Ready Software was involved with USAF prediction model development • Several factors identified by USAF were incorporated into model • Model works for any software intensive engineering system • Continually verified against new data sets • Continually updated for new development processes • 40% of data is from defense, aerospace and space • 30% is from medical devices • 30% is from transportation, energy, major equipment • All data is from mission/safety critical systems

Mission Ready Software, 2022

History of predictive models – one bad apple can spoil the whole bunch

- RIAC 217 Keene model was developed by personnel who don't have real software engineering experience
- The model had several technical statistical errors and faulty assumptions
  - Assumed that software reliability would grow for 4 years
  - Neglected to consider that no one on earth waits 4 years for a new feature release
  - Any time new features are added to code, the defect profile resets
  - Many faulty assumptions about the CMMi level and defects
- The model was very popular with reliability engineers because it always yielded MTBF predictions of virtually infinite because of the 4 years of growth
- The customers know the model is grossly incorrect and hence were cold to prediction models
- However, the IEEE 1633 Recommended Practices for Software Reliability, 2016 improved on that perception by listing the models to NOT use and why the models should not be used
- **Mission Ready Software developed a model based on facts and not opinions**

Mission Ready Software Predictions

- We have > 9 times the data that USAF benchmarked
- We have > 7300 times the data of the Keene Cole Model
- We have the only machine learning model for predicting defects in software before the code is even written
- We keep the model up to date every year for
  - Emerging development practices such as DevSecOps, AI, ML, etc.
  - More data sets from industry
  - More application types (ie driverless vehicles, hand held medical devices, etc.)

Factors that have been proven to be related to software reliability

- To date Mission Ready Software has correlated 679 factors to actual escaped defect density
  - 156 factors were not employed by enough organizations to be usable in a predictive model. The remaining 523 factors are summarized as shown below.

Characteristic category	Number /% of characteristics in this category	Examples of characteristics in this category
Product	50 – (10%)	Size, complexity, whether the design is object oriented, whether the requirements are consistent, code that is old and fragile, etc.
Product risks	12 – (2%)	Risks imposed by end users, government regulations, customers, product maturity, etc.
People	38 – (7%)	Turnover, geographical location, amount of noise in work area, number of years of experience in the applicable industry, number of software people, ratio of software developers to testers, etc.
Process	121 – (23%)	Procedures, compliance, exit criteria, standards, etc.
Technique	302 – (58%)	The specific methods, approaches and tools that are used to develop the software. Example: Using a SFMEA to help identify the exceptions that should be designed and coded.

assessment

Static analysis tools measure these

These are often overlooked

SEI CMMi assesses this

These are often overlooked

Comparison of factors that have been quantitatively correlated to reduced defects

	CMMi Assessment	Rome Laboratory Model	Requs AI Predict
End user domain experience of software engineers	No	Yes	Yes
Level of rigor of testing, HWIL, fault injection testing, etc.	No	Yes	Yes
Visual representations such as diagrams, tables, etc.	No	Yes	Yes
Shorter release cycles	No	No	Yes
Processes to control source code, changes and versions	Yes	No	Yes
Written specifications, design, test procedures	Yes	Yes	Yes
Regular reviews with software engineers, schedules granular to day or week	No	No	Yes
Quality processes	Yes	Yes	Yes
Coding practices	No	Yes	Yes
Unit testing level of rigor	No	Partial	Yes
Location of software engineers with respect to system hardware and other engineers and each other; remote working, etc.	No	No	Yes
Use of advanced life cycle models (CD/CI), stakeholder involvement	No	No	Yes
Object oriented methods	No	No	Yes
Avoidance of too many inherent technical risks in one release	No	No	Yes
Ability to accurately schedule the work	No	No	Yes

Mission Ready Software, 2022

Mission Ready Software Statistics for various SRE capabilities

Cluster	Outcome	Defect metrics			Late deliveries (as per SW estimates)	
		Average defects per 1000 source lines of code	% defects removed prior to release	Fault rate	Prob (late)	How much project is late by as % of schedule
3%	World Class	.0269	>75%	Steadily decreasing	40	12
10%	Successful	.0644			20	25
25%	Above average	.111	40-75%	Recently peaked or recently decreasing	17	25
50%	Average	.239			34	37
75%	Below average	.647			85	125
90%	Impaired	1.119	<40%	Increasing or peaking	67	67
97%	Distressed	2.402			83	75

- Actual defect density from 150+ software/firmware projects in Mission Ready Software database benchmarked into one of these 7 clusters
- Organizations with lowest deployed defect density were also late less often and by a smaller amount
- SRE for any given project can be predicted by answering a simple survey
- Average defects/1000 SLOC calculated by adding all defects over life of version (2-8 years) and dividing by the actual size of that version

Mission Ready Software Predictions - Accuracy

- We continually update the model by validating it against new data sets
- We also validate the model results against the "Subject Matter Expert (SME)" guesses
  - Our models are within 1 order of magnitude when used properly
  - SME Guesses are almost always 5+ orders of magnitude
- If the model predicts the cluster correctly - the relative error is negligible
- If the model predicts one cluster in either direction - the error is typically < 1 OOM
- If the model predicts Low, Medium or High accurately - the error is typically at 1 OOM
- The model is least accurate for software organizations that pretend to be world class but really are not
  - We are working on finding patterns to identify these as well
- The model is very accurate at identifying distressed programs

PRACTICAL APPLICATIONS OF THE PREDICTIONS

- Predicting defect pileup before it kills the schedule
- Planning resources to test, fix defects and provide field support
- Predicting overall success or failure of release
- Sensitivity analysis

PREDICTING DEFECT PILEUP BEFORE IT KILLS THE SCHEDULE

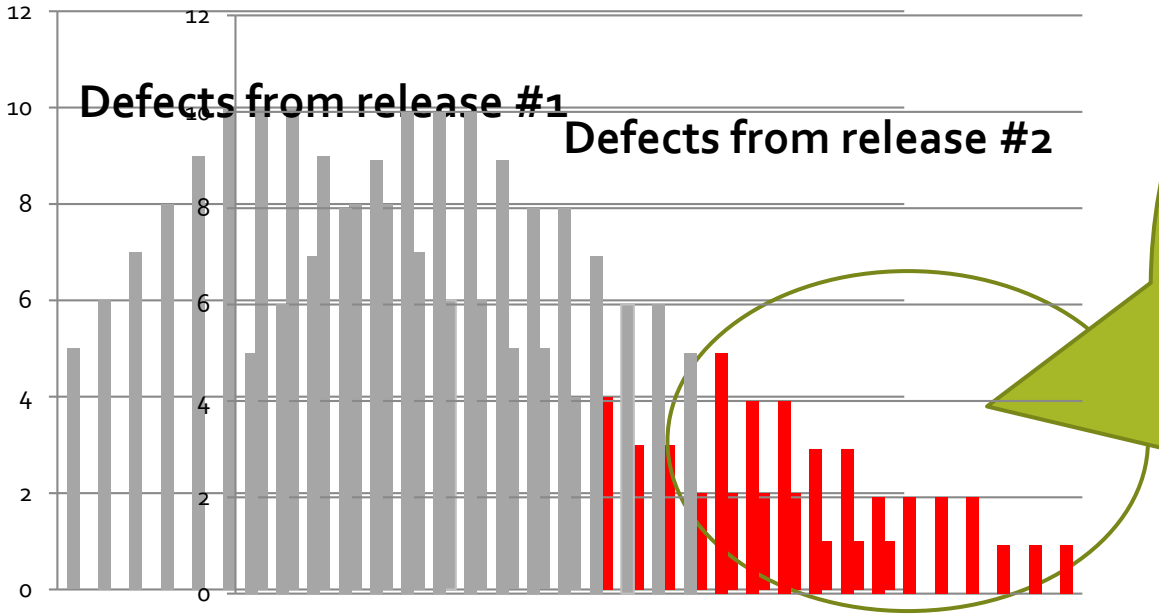
Defect pileup creeps up on software engineering

Defect pileup eventually will cause the entire schedule to slip because people are redirected to fixing defects instead of working on future features

The models can see it coming - well before it causes schedule delay

Predictions can be used to plan release sizes/frequency to avoid defect pileup

- Software is never deployed with just one big software release even with waterfall model
- Superimpose predicted defect profile from every release onto one timeline
- Are there obvious signs of defect pileup?



Red – Predicted to be found by customer
Grey – Predicted to be found by developer

In this example, defects are piling up from release to release
Solutions to pileup –
1) Split features up into more smaller releases
2) Keep the same spacing but less new code in each release
3) Keep the same code size but greater spacing.

MISSION READY SOFTWARE
SOFTREL LLC

PLANNING RESOURCES

Number of test personnel

Corrective action

Field support

Resource planning

- Model predicts defects to be found in test and in operation
- Staffing effort = typical corrective action time & defects
- Model predicts when the defects will be discovered so that resources can be scheduled
  - *Most defects are found in the first year of operation*
- Model predicts test hours needed to find the defects

Week of testing	Total defects predicted this week	Total critical defects predicted for this week	Average # people needed to address all defects	Average # people needed to address critical defects
Week 1	12.909	1.033	5.2	0.4
Week 2	11.800	0.944	4.7	0.4
Week 3	10.787	0.863	4.3	0.3
Week 4	9.860	0.789	3.9	0.3
Week 5	9.013	0.721	3.6	0.3
Week 6	8.239	0.659	3.3	0.3

Hours after delivery	Total defects predicted to be found this increment
700	14.9
1400	14.1
2100	13.3
2800	12.6
3500	11.9
4200	14.0
4900	13.2
5600	12.4
6300	11.7
7000	11.1
7700	10.5
8400	12.6

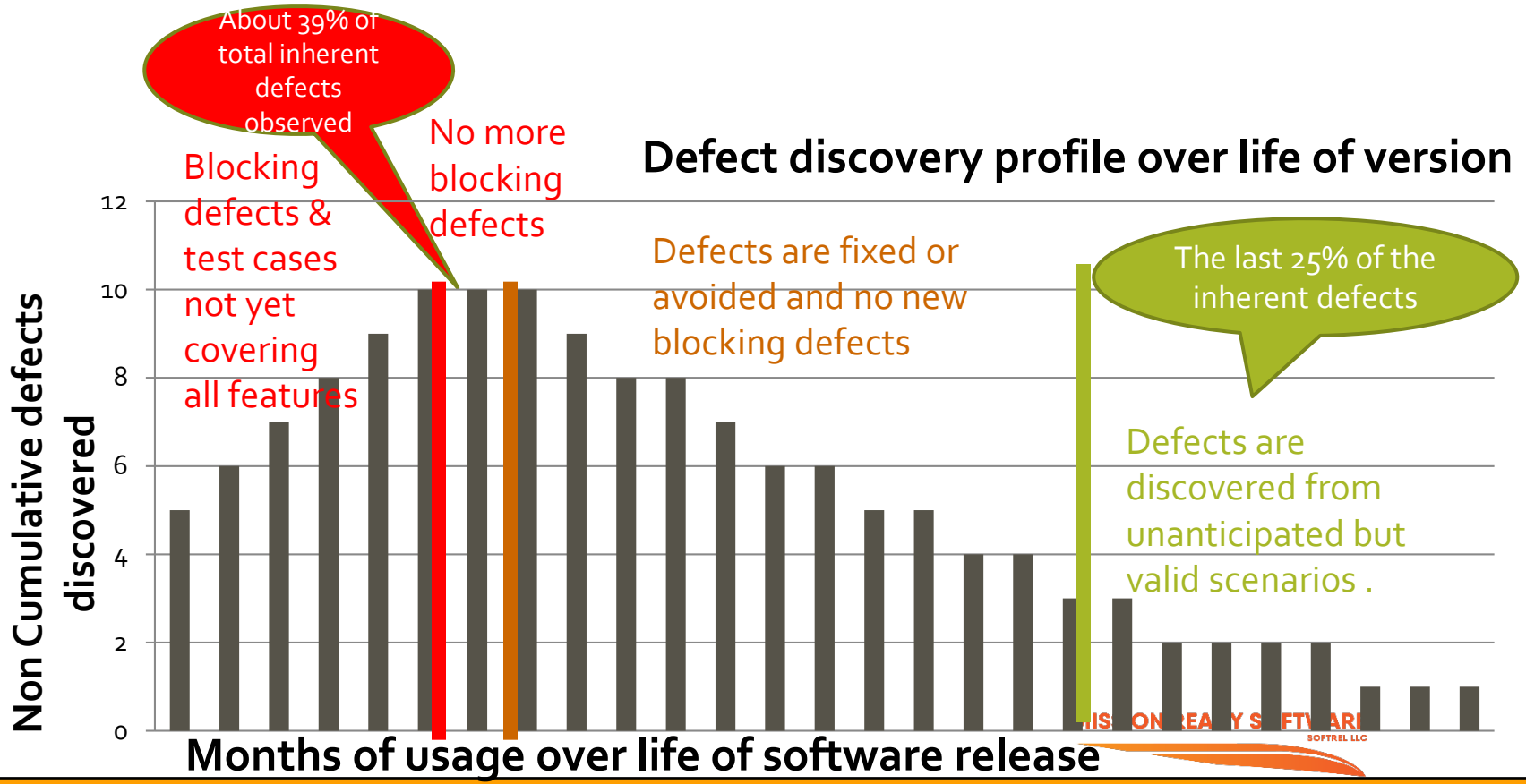
PREDICTING OVERALL SUCCESS

Successful, Mediocre, Distressed

Defect Removal Efficiency – Percentage of total inherent defects that have been observed so far in test or usage

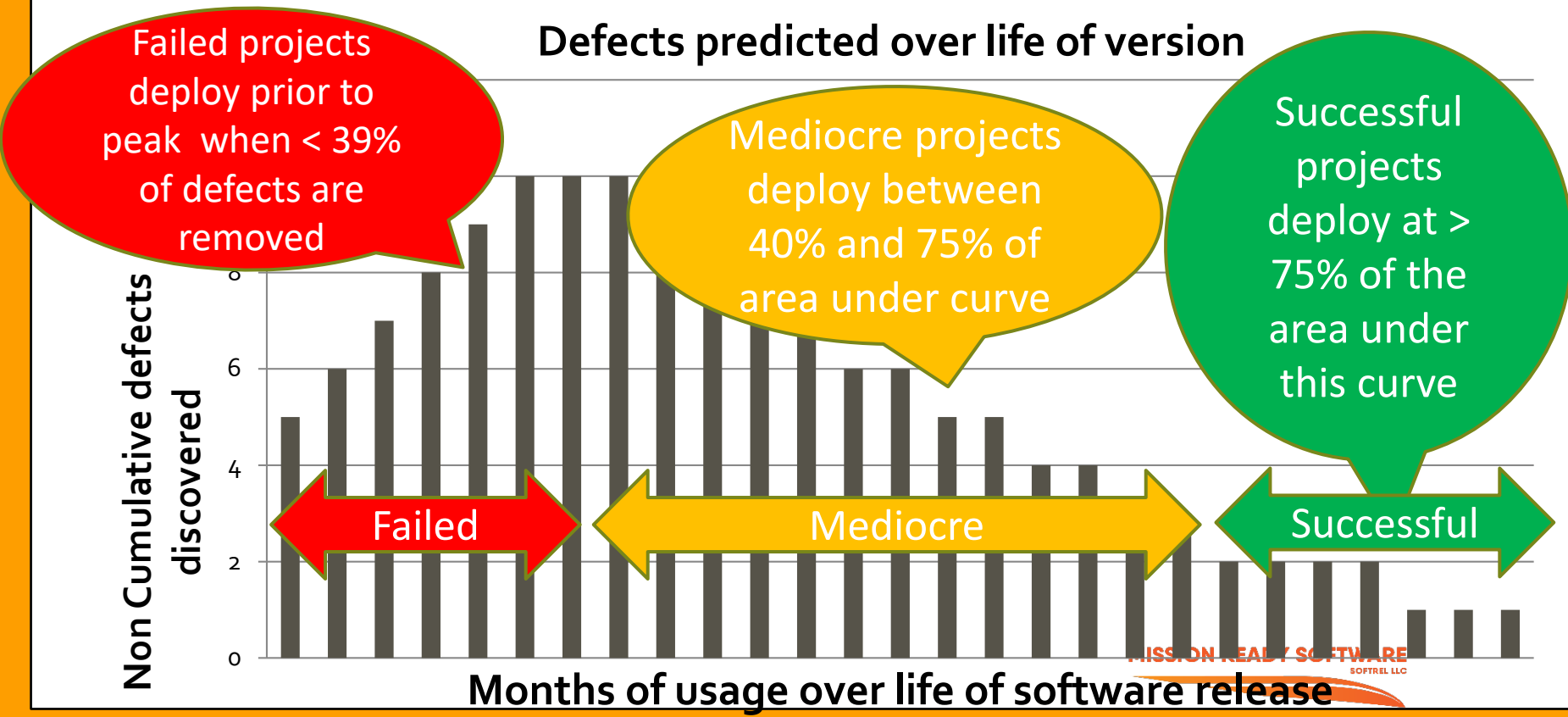
Defect discovery profiles

- Since the 1970s it has been known that for any release (waterfall or agile) that defects found per usage hours will increase, peak, decrease and trail off
- Increase is due to blocking defects and the fact that every feature cannot be tested at the same time
- Peak happens when the blocking defects are fixed or avoided and there's no more blocking defects
- If the defects are fixed or avoided then defect discovery rate will start to decline during testing
- The tail of the Rayleigh curve can last years. This is where a few defects due to unanticipated but valid conditions are discovered.



Four things that vary from SW project to project

1. Height of the Rayleigh curve – more software features means more defects
2. Width of the Rayleigh curve – depends on how many different users operate the software (From 1 year for mass deployed software to 4+ years for non mass produced)
3. When will the software be deployed with respect to this profile. Failed? Mediocre? Successful?
4. The spacing between the releases. With incremental/agile development the spacing is much closer and the size of each curve is much smaller than with waterfall development.



SENSITIVITY ANALYSIS

Models can be used to identify the fewest lowest cost changes in development practices with biggest return on investment (ROI)

Models can also be used to identify the development factors THAT DON'T REDUCE DEFECTS AS MUCH AS PEOPLE THINK

Every hour you spend on practices that aren't effective is an hour not spent on practices that are!

Once the prediction is established, alternative scenarios are identified by Requs AI Predict

The "gaps" are the development practices that you aren't doing but are most likely to reduce defects

- Relative culture change required to implement the practice or method
- Relative out of pocket cost required to implement the practice or method (i.e. hiring people, buying tools, etc.)
- Relative number of release cycles until you see visible improvement
- Prerequisites needed to attempt the development practice
- All development factors are listed in ranked order of impact on defect reduction

The "excess" factors are those practices that you are doing that have limited ROI

- Example: An organization has 100% affirmative responses for software process but few affirmative responses

The worst possible scenario is that an organization invests a lot of money and time into a development practice that has only marginal effect on reducing defects

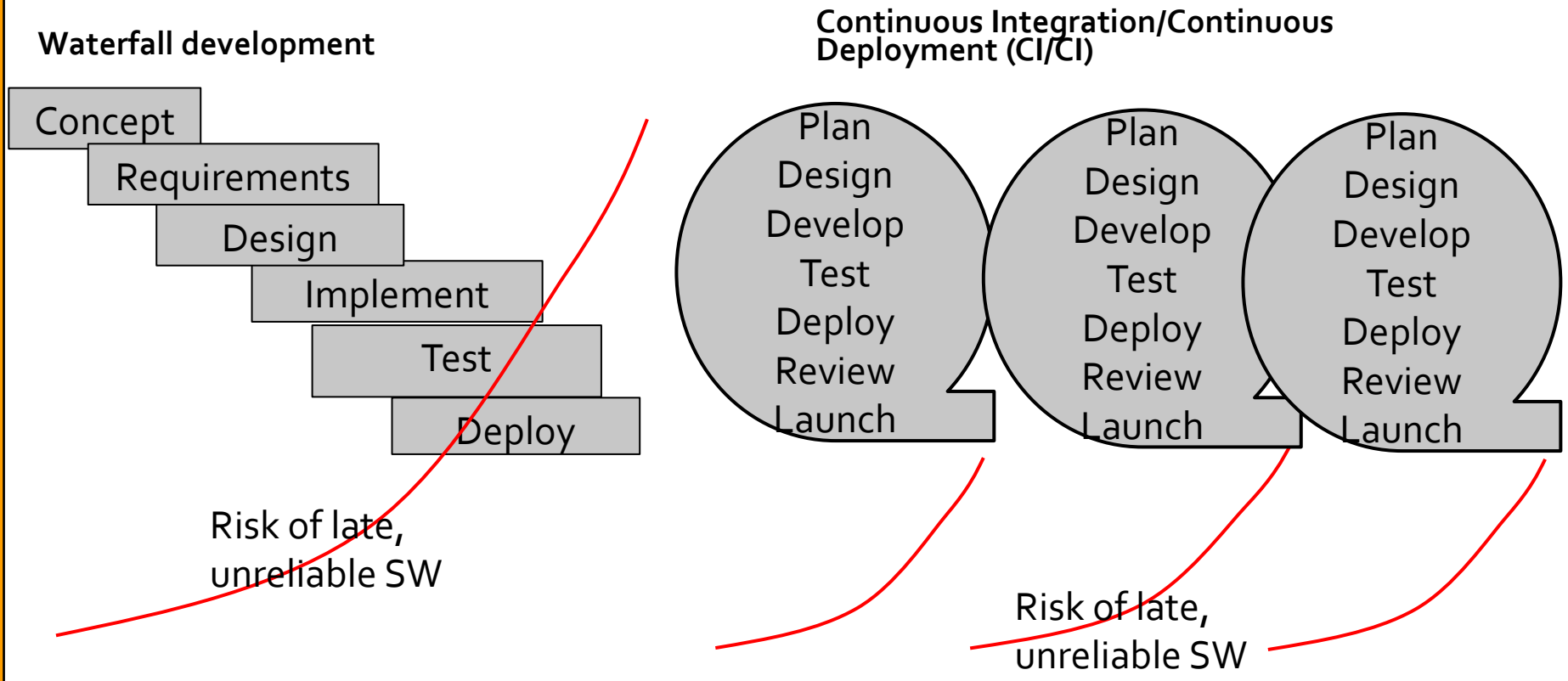
Our tool helps you to avoid popular silver bullets

Sensitivity analysis

RELIABLE SOFTWARE AND CONTINUOUS INTEGRATION/ CONTINUOUS DEVELOPMENT

Not every risk is mitigated by CI/CD

Waterfall development versus CI/CD



- Long cycle times and "big blobs" typically result in late and unreliable SW

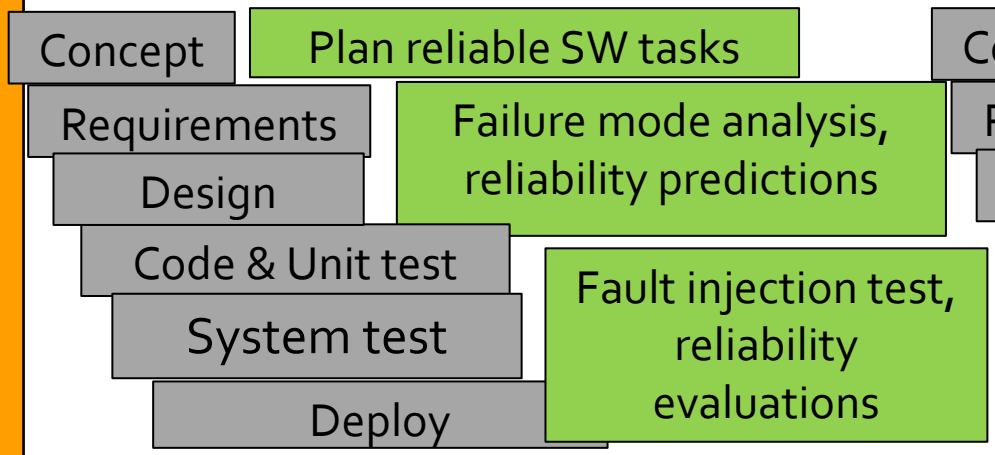
- Smaller cycle times reduce some *but not all* risks

- Low test coverage or level of rigor in testing
- Not enough defects are fixed in each sprint - so they pile up to the next sprint
- Software engineers (SE) don't have end user/industry knowledge
- SEs misunderstand the user stories (largely due to lack of end user experience)
- SEs skip design or aren't good at it
- SEs don't test software in real world environment
- SEs don't do consistent unit testing against design or specifications
- SEs don't consider all failure modes or scenarios
- SEs aren't good at estimating how much work they can do in a sprint (which leads to late deliveries which is never good for reliability)
- Despite the fact that CD/CI was invented for this purpose- SEs don't take advantage of data from each sprint so as to plan/replan the scope and effort for future sprints
- SEs sometimes tag "Agile" to anything they conveniently do or do not want to do
- SEs typically want development tasks to be "Agile" but reliability tasks to be "Waterfall"

Risks that aren't necessarily mitigated by CI/CD

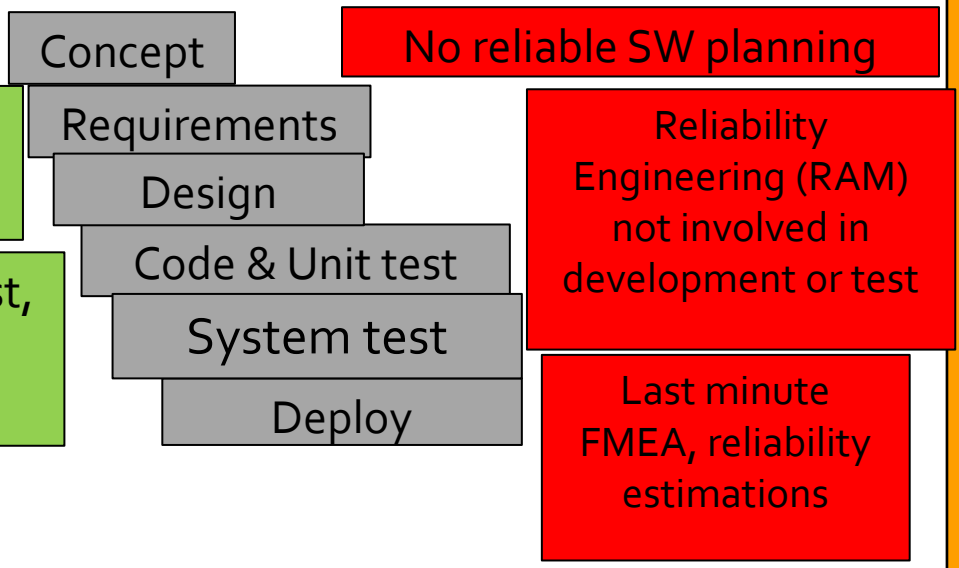
Ideal versus real world relationship between reliability and software engineering

Waterfall As per IEEE 1633



- Reliable SW tasks are supposed to be in line with development

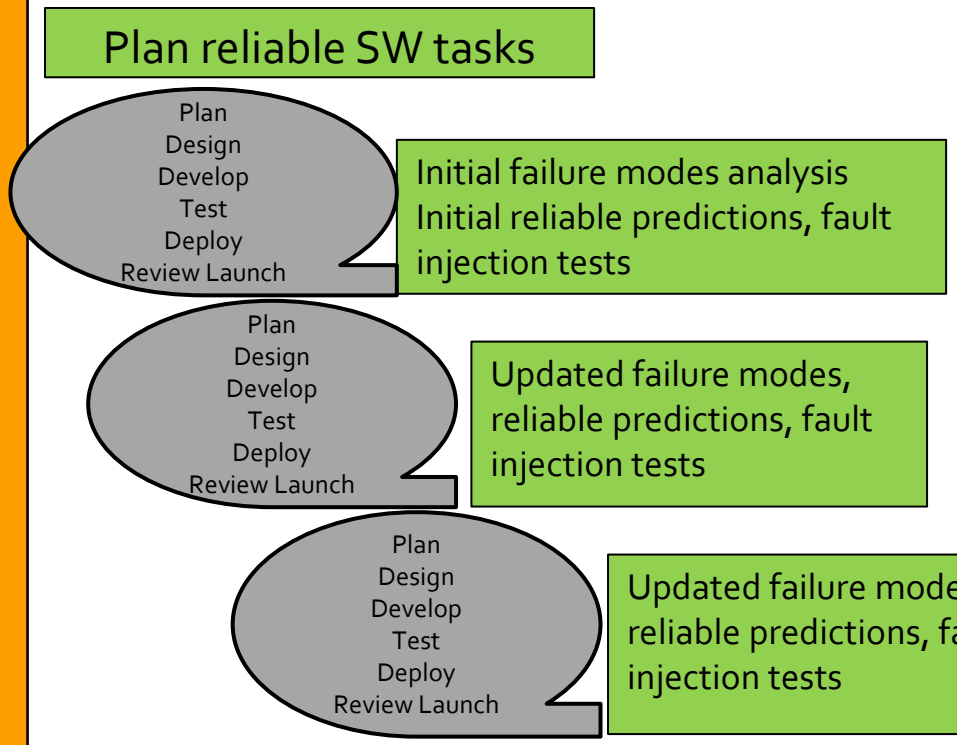
What really happens



- Reliable SW tasks are done after project is already in trouble and no time to fix anything. Everyone is blind sided by poor reliability.

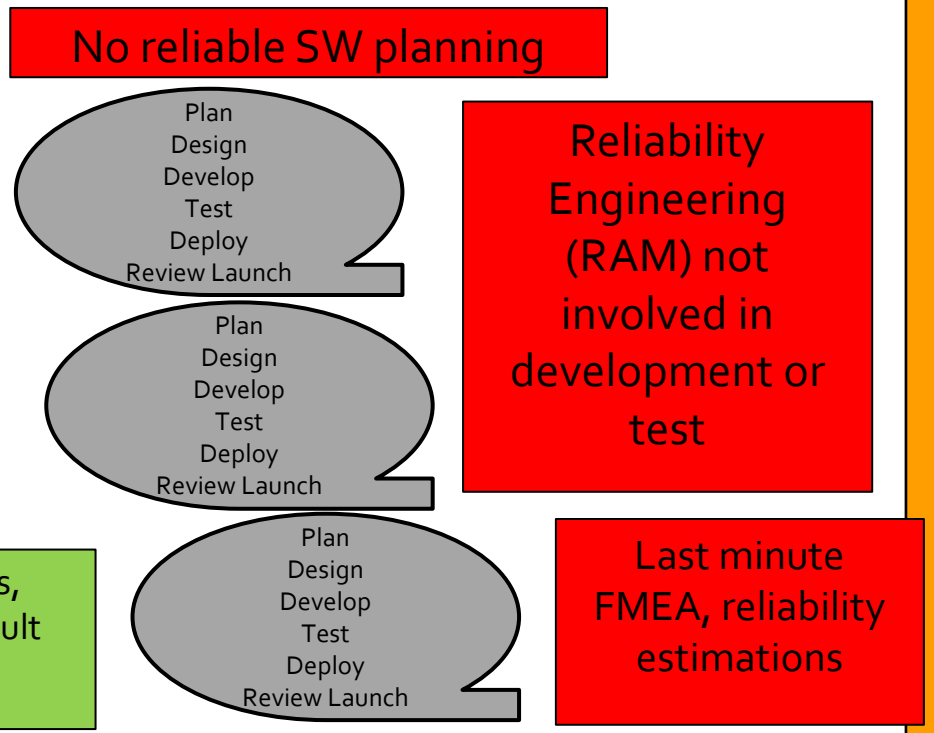
Ideal versus real world relationship between reliability and software engineering

CI/CD as per IEEE 1633



- Reliable SW tasks are supposed to be integrated with CD/CI

What really happens



- Reliable SW tasks are done after project is already in trouble and no time to fix anything. Everyone is blind sided by poor reliability.

Root causes for late RAM involvement

Software engineering (SE) doesn't want RAM involved

- Justified reasons
  - RAM engineers may neglect to read the software specifications, user manuals, test plans or procedures, etc., to familiarize themselves with the SW
  - RAM engineers try to apply HW reliability concepts that don't work for SW
  - RAM engineers don't understand the failure modes or how to identify them
- Unjustified reasons
  - SE doesn't really want to fix defects, test fault injection or know how reliable the software is
  - SE thinks/says that IEEE 1633 doesn't apply to CD/CI (which is not true)

RAM engineers don't want early involvement

- Justified reasons
  - RAM engineers are often excluded from scrum teams
- Unjustified reasons
  - RAM engineers grossly underestimate the amount of SW in the system
  - RAM engineers assume that the defects can be easily fixed

BACKUP

The Agile Manifesto

Common CI/CD Myths

The Agile Manifesto

Written in 2001 by:

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

To address the many problems with Waterfall software development

https://agilemanifesto.org/

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Since 2001 there have been many adaptions that have muddied the original vision for Agile development

- *"Unfortunately, once a movement becomes popular, the name of that movement gets blurred through misunderstanding and usurpation. Products and methods having nothing to do with the actual movement will borrow the name to cash in on the name's popularity and significance. And so it has been with Agile."*
- Robert C Martin, "Clean Agile".
- Robert is one of the original members of the Agile Manifesto

Myth #1 – We will have more reliable on time software because we are using Agile.

Myth #2 – We don't do software sizing with Agile. (Sizing is estimating people or effort or time to develop the software)

Myth #1 - Statement from Robert C Martin:

"Agile is a framework that helps developers and managers execute pragmatic project management. However, management is not made automatic and there is no guarantee that managers will make appropriate decisions. Indeed it is entirely possible to within the Agile framework and still completely mismanage the project and drive it to failure."

Myth #2- Statement from Robert C Martin with regards to the analysis phase of development:

"Of course, some things are obvious. We should be sizing the project and doing basic feasibility and human resource projections. That is the least that our business would be expecting of us. "

This ties to the Agile Manifesto "*At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."*

Myth #3 –We estimate story points that are associated with relative effort. However, we don't try to estimate the remaining effort in terms of absolute values. i.e. how many months or people we need to finish X functionality.

It's impossible to estimate that!!!

Statement from Robert C Martin*:*

Mr. Martin illustrates:

- *Two charts showing 1) actual velocity of story points over time and 2) Burn down chart of estimated story points remaining;*
- *The Iron Cross - which consists of Good, Fast, Cheap and Finished. As per Mr. Martin you can only have 3 of the 4 attributes of the iron cross.*

"Each of the attributes has coefficients. A good manager manages coefficients on these attributes rather than demanding that all be 100%. It is this kind of management that Agile strives to enable.

It is critical goal of Agile to get those two charts on the wall. One of the driving motivations for Agile software development is to provide the data that managers need to decide how to set the coefficients on the Iron Cross and drive the project to the best possible outcome."

"Iteration one begins with an estimate of how many user stories will be completed…At the end of the iteration, some fraction of the stories that we had planned to finish will be done. This gives us our first measurement of how much can be completed in an iteration. This is real data. If we assume that every iteration will be similar, then we can use that data to adjust our original plan and calculate a new end date for the project"

Myth #4 –We can't have fast development and have reliable software.

Statement from Robert C Martin:

"Everyone knows that you can go much faster by producing crap. So stop writing all those tests, stop doing all those code reviews, stop all that refactoring nonsense and just code….

I'm sure you know that I am going to tell you that this is futile. Producing crap does not make you go faster, it makes you go slower. This is the lesson you learn after you've been a programmer for 20 or 30 years. There is no such thing as quick and dirty. Anything dirty is slow."

"The only way to go fast is to go well. "

Facts from Mission Ready Software 30 year benchmarking study of almost 200 engineering companies proves that this isn't true.

The engineering companies that develop reliable software (shown in green) were also on time more often. When they were late – the slip amount was much less than the organizations that cut corners.

Organizations that deployed the software when the fault rate was increasing also very late by a non-trivial amount. In addition, their customers considered the project to be a failure.

		Defect metrics			Late deliveries (as per SW estimates)	
Cluster	Outcome from customer perspective	Average defect density	% defects removed prior to release	Fault rate	Prob (late)	How much project is late by as % of schedule
3%	World Class	.0269	>75%	Steadily decreasing	40	12
10%	Successful	.0644			20	25
25%	Above average	.111	40-75%	Recently peaked or recently decreasing	17	25
50%	Average	.239			34	37
75%	Below average	.647			85	125
90%	Impaired	1.119	<40%	Increasing or peaking	67	67
97%	Distressed	2.402			83	75

The majority of these organizations are using agile development. Very few are still using Waterfall model.

Myth #5 – We don't need requirements or tracing of the requirements.

Myth #6 – We don't need to analyze failure modes.

Myth #7 – We do developer unit testing only.

The Agile Manifesto:

"The best architectures, requirements, and designs emerge from self-organizing teams."

The manifesto makes no mention of neglecting the requirements.

Statement from Robert C Martin:

"A specification is, by its very nature, a test.

For example: When the user enters a valid username and password, and then clicks "login" the system will present the "Welcome" page.

This is obviously a specification. It is also obviously a test….This is the practice of Acceptance tests".

"Acceptance tests are a collaborative effort between business analysts, QA and the developers. Business analysis specify the happy paths… QA's role is to write the unhappy paths. There are a lot more of them then there are the former .

QA folks are hired for their ability o figure out how to break the system. They are deeply technical people who can foresee all of the strange and bizarre things that users are going to do to the system.. … And of course, the developers work with the QA and business analysts to ensure that the tests make sense from a technical point of view."

Conclusion: You can't develop the acceptance tests without somehow keeping track of the user's requirements. Clearly the tests are from multiple points of view. The unhappy paths are captured in the "Common Defect Enumeration".

Common Defect Enumeration used for identifying the unhappy paths as well as the failure modes effects analysis

The Common Defect Enumeration (CDE) [2], developed by Ann Marie Neufelder, is derived from 60 years of "*all of the strange and bizarre things that users are going to do to the system".* The categories include:

- Faulty state management
- Faulty error handling
- Faulty processing
- Faulty functionality
- Faulty data
- Faulty timing
- Faulty sequencing
- Faulty usability
- Faulty machine learning
- Faulty interfaces

The CDE will be published on the DAU R&M CoP website shortly.

User requirements tracing

- These industries are required to identify which user requirements are covered
  - Medical devices
  - Vehicles
  - Commercial aviation
- All of these industries are using Agile development

Myth #8 We don't need to do design because we are using Agile.

The Agile Manifesto:

"The best architectures, requirements, and designs emerge from self-organizing teams."

"Continuous attention to technical excellence and good design enhances agility."

Clearly the writers of the manifesto intended for software engineers to do software design.

Robert C Martin mentions design regularly throughout his book.

- As part of Agile – 24,25
- Test Drive Design – 121
- Simple design – page 125
- Design weight – 127

So clearly it is intended to be part of Agile.

Myth #9 – We have to have 2 week engineering cycles.

- Our benchmarking study shows that all software project that exceeds 18 months of development are distressed.
- This confirms the Agile Manifesto "*Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale*"
- However, our data [1] shows that any project that has cycle time <= 3 months was either average or above average in on time delivery and reliability.
- We have the world's largest benchmarking study[1] and see no physical evidence that either on time delivery or reliability is better with 2 week cycles.
- However, our data [1] clearly shows that when software engineers have regular meetings with subject matter experts either daily or weekly that their software is more reliable than when they meet less often.
- It is our conclusion that the daily/weekly meetings are why the Agile Manifesto prefers the shorter timescale.

Improvements to rectify last minute un-reliable software

Obstacle	Improvement areas
SE think/say reliability doesn't apply to CD/CI	IEEE 1633 2016 provided clear guidance. The 2023 update will make it excruciatingly obvious. Mission Ready Software provides training on how to apply reliability in CD/CI.
SE doesn't allow reliability into sprint decision making, failure mode identification or fault injection testing	Requires a good SOW and clear company and industry standards. Mission Ready Software has rewritten the SAE and IEEE and DoD guidance for applying Reliable Software in CD/CI.
SE assumes CD/CI fixes all risks	Risks can be predicted with reliable software prediction models such as Requs AI Predict and failure mode analysis using our Common Defect Enumerations.
RAM assumes software is small part of system	IEEE 1633 has methods for predicting the portion of SW versus HW failures. RAM engineers need to start using this guide. Mission Ready Software provides training and templates for this prediction.
RAM assumes defects can be fixed easily	IEEE 1633 will be updated to make it much clearer to RAM people that reliability cannot start the last 30 days of the program. Our training classes provide clear guidance.
RAM engineers try to apply HW reliability concepts to SW that don't work	• IEEE 1633 has guidance but 2023 update will provide crystal clear examples of effective versus ineffective software FMEAs, and Common Defect Enumerations. • Common Defect Enumerations have been published on DAU R&M CoP website.

References

- [1] AM Neufelder, "The Cold Hard Truth About Software Reliability", Version 6i, 2019.
- https://agilemanifesto.org
- Robert C. Martin, "Clean Agile – Back To Basics", Pearson Publishing, 2020.
- [2] AM Neufelder, "The Common Defect Enumeration", 2020.