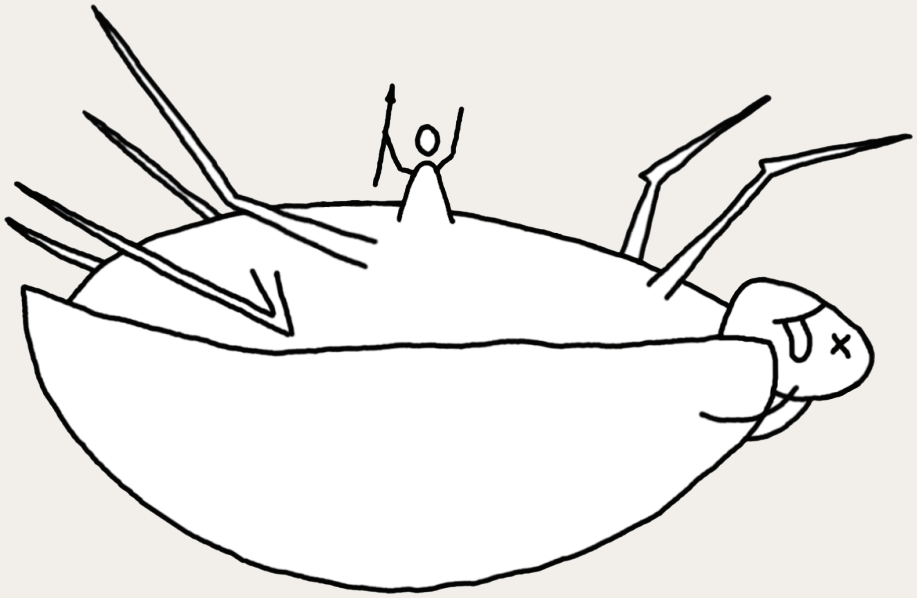
bbv
MAKING VISIONS WORK.



PROJECT INSIGHTS

ZERO BUG POLICY

EIN ERFAHRUNGSBERICHT

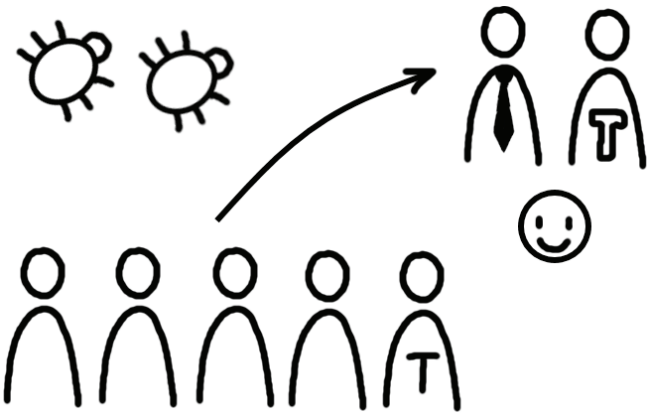
Ein fremder Planet. Ein Aussenposten der Menschen. Alles ist ruhig. Plötzlich schreit jemand: «Bugs! Bugs!» Soldaten rennen auf die Verteidigungsmauer. Die Kamera folgt ihnen und zeigt den Zuschauern den Blick auf eine weite Ebene. In der Ferne hinter einer Hügelkuppe erscheint die erste Angriffswelle der Ausserirdischen. Es sind riesige Käfermonster. Bugs.

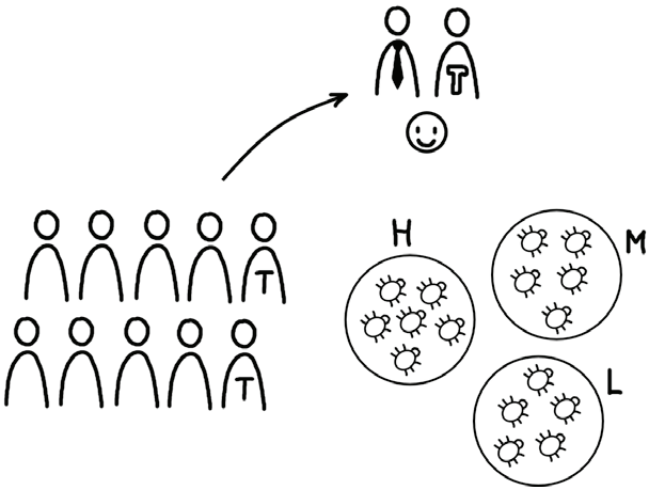
Zuerst sind es nur wenige, aber dann folgen immer mehr. Hunderte. Tausende. Die Soldaten feuern in die Menge der Bugs, aber es sind einfach zu viele. Die Bugs erreichen die Mauer, klettern übereinander hinweg und überrennen schliesslich die Soldaten.

Dies ist eine Szene aus einem alten Science-Fiction-Film[1]. Für mich ist es leider aber auch eine Szene aus einem vergangenen Softwareprojekt.

DER STURM ZIEHT AUF

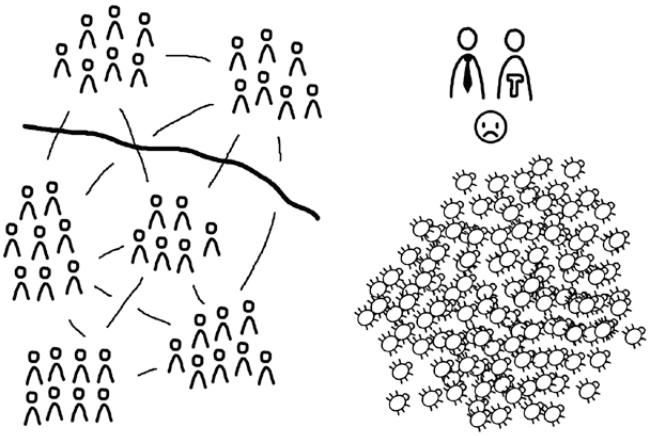
Es begann alles ganz harmlos. Wir waren ein kleines Scrum-Team mit einem eigenen Testspezialisten. Bei unserem Projekt handelte es sich um eine Geräteentwicklung, bei der wir leider keinen direkten Kundenkontakt hatten. Die Ergebnisse unserer Sprints lieferten wir darum an eine firmeninterne Testabteilung. Zu diesem Zeitpunkt hatten wir die Bugs im Griff, und unsere Abnehmer waren sehr zufrieden mit der gelieferten Qualität.





Dann kam ein zweites Team dazu. Wir lieferten zwar immer noch gute Qualität, produzierten aber auch mehr Bugs. Um die Planung zu vereinfachen, bekamen die Bugs nun Prioritäten: hoch, mittel oder tief. «Hoch» priorisierte Bugs erledigten wir sofort. Bugs mit der Priorität «Mittel» kamen an die Reihe, wenn wir etwas Extrazeit hatten (was kaum der Fall war). Die restlichen Bugs erschienen gar nicht wirklich auf unserem Radar.

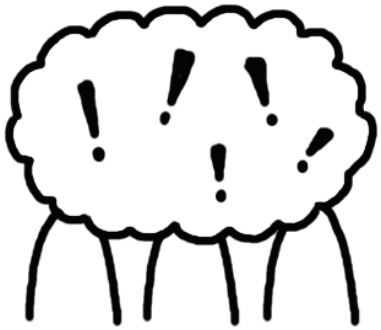
Während des Projekts gab es unerwartet ein umfangreiches Paket mit neuen Anforderungen. Um Verzögerungen zu vermeiden, erhielten wir Unterstützung von vier weiteren Teams. Zwei davon arbeiteten an einem anderen Standort im Ausland. Die Architektur war aber immer noch auf ein Team ausgelegt. Und so standen sich alle gegenseitig auf den Füssen und die Anzahl Bugs wurde schnell dreistellig.



ÜBERRANNT

Die neue Situation brachte viele Probleme mit sich. Die überwältigende Anzahl an Bugs führte zu einer gewissen Hoffnungslosigkeit in den Teams. Was kümmerte uns ein neuer Bug? Es war ja nur einer von vielen.

Statt die Bugs zu beheben, verbrachten wir viel Zeit damit, um die Bugs herumzuarbeiten. So lief die Vorbereitung der Demo einer neuen Funktion typischerweise wie folgt ab:



«Software auf Zielsystem installieren … alle Daten für die Demo vorbereiten … Funktion zur Sicherheit nochmals testen … Mist, funktioniert nicht … wieso … hm … wenn ich das versuche … nein … und das … nein … ich frage mal die anderen … ah, das ist der bekannte Bug X … um den zu umgehen, muss ich vor der Installation eine Datei manuell anpassen … also alles nochmals von vorn … super, jetzt funktioniert es … nochmals neu installieren, damit die Demo auf einem sauberen System stattfindet … alle Daten für die Demo vorbereiten … fertig.»

Etwas später während der Demo:
«Funktion demonstrieren … Mist, funktioniert schon wieder nicht … oh, habe beim zweiten Mal Installieren vergessen, die Datei anzupassen … also alles nochmals von vorn … seufz.»

Auch mit dem Bug Management verbrachten wir viel Zeit. Einmal pro Woche wurden die Bugs priorisiert und den Teams zugewiesen. Folgender Ablauf war nicht ungewöhnlich:

1. Woche: Bug-Management-Team bespricht Bug X. Das Team versteht die Beschreibung nicht und schickt eine entsprechende Anfrage an den Erfasser.

2. Woche: Bug-Management-Team bespricht Bug X mit dem Feedback des Erfassers und entscheidet sich, Team A den Bug zuzuweisen.

3. Woche: Team A schaut sich Bug X an. Es erkennt, dass die Zuweisung falsch ist und schickt ihn zurück ans Bug-Management-Team.

4. Woche: Bug-Management-Team bespricht Bug X. Irgendwie kommt er dem Team bekannt vor. Mit einem unangenehmen Gefühl von Déjà-vu weist es den Bug Team A zu.

5. Woche: Team A schaut sich Bug X an. Es e rkennt, dass die Zuweisung falsch ist und redet diesmal direkt mit jemandem vom Bug-Management-Team.

6. Woche: Bug-Management-Team bespricht Bug X und weist ihn Team B zu.

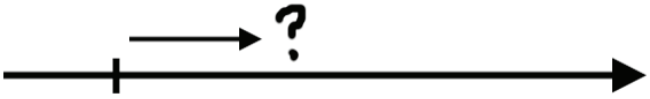
7. Woche: usw.

Solches Bug-Management und die Priorisierung von Bugs führten dazu, dass manche Bugs sehr alt wurden, bis sie zu einem Entwickler gelangten. Hatte man dann Fragen zu dem Bug, konnte sich der Erfasser üblicherweise kaum mehr an Details erinnern.

War der Bug in der aktuellen Version nicht mehr reproduzierbar, stand man vor einer schwierigen Entscheidung: Annehmen, dass der Bug inzwischen durch Änderungen am Code behoben wurde, oder auf die alte Version zurückgehen, um den Bug dort zu reproduzieren?

Bei solchen Bugs konnte es sein, dass man auf dem Zielsystem zuerst ein altes Image aufspielen oder sogar eine alte Hardware aus dem Schrank holen musste.

Auch für die Projektplanung war die Situation kein Spass. Bugs waren kaum planbar. Vermeintlich simple Fehler entpuppten sich als wahre Monster, während scheinbare Monster ganz einfach zu bewältigen waren. Es war beinahe unmöglich abzuschätzen, bis wann das Produkt fertiggestellt sein würde.



ZERO BUG POLICY

Niemand war glücklich. Wir alle wollten etwas an der Situation ändern. Aber was? Da fiel jemandem eine Aussage ein, die er an einer Scrum-Master-Zertifizierung gehört hatte: «Bei uns überlebt kein Bug die Nacht.» Das klang ziemlich abenteuerlich, aber was hatten wir schon zu verlieren?

Also versuchten wir, die Projektleitung von einer Zero Bug Policy zu überzeugen. Wir wollten nicht länger an neuen Funktionen arbeiten, solange es noch kritische Bugs im System gab. Und wir hatten Glück. Am Ende war die Projektleitung an Bord und stand voll hinter uns.

Damals legten wir einfach los. Hätten wir uns nur etwas Zeit genommen und nach bereits bestehenden Lösungen gesucht, dann hätten wir einen grossen Rückschlag verhindern können. Aber ein Schritt nach dem anderen. Hier zuerst die Definition, die uns gefehlt hat.

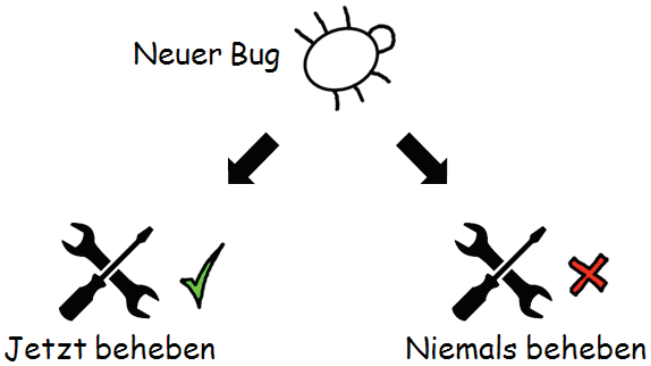
Zero Bug Policy
Keine bekannten Bugs

Aufgepasst: nicht keine Bugs, sondern keine *bekannten* Bugs!

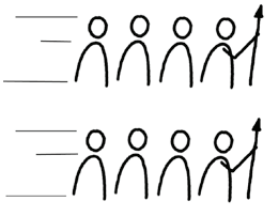
Die Zero Bug Policy ist auch als *Zero Bug Tolerance* bekannt. Das führt häufig zu dem Missverständnis, dass man ein komplett fehlerfreies Produkt erstellen will. Ab einer gewissen Komplexität ist das aber nicht mehr wirtschaftlich. Die Zero Bug Policy betrifft nur bekannte Bugs. Sie sagt nichts darüber aus, wie viel Aufwand in weitere Bug-Suche investiert werden soll, wenn zu einem gewissen Zeitpunkt keine bekannten Bugs mehr existieren.

Aber heisst das nun, dass *alle* bekannten Bugs behoben werden müssen? Ja und nein. Die Zero Bug Policy löst dieses Problem sehr simpel. Wenn ein neuer Bug entdeckt wird, gibt es nur zwei Möglichkeiten:

- Der Bug wird sofort behoben.
- Der Bug wird nicht behoben.



Und um Missverständnisse zu vermeiden: Entscheidet man sich, den Bug nicht zu beheben, gibt es auch kein Management oder Tracking für diesen Bug. Es ist per Definition kein Bug mehr, sondern ein akzeptiertes Verhalten des Produkts. Das klingt radikal, ist aber notwendig, damit die Zero Bug Policy ihr ganzes Potenzial entfalten kann.

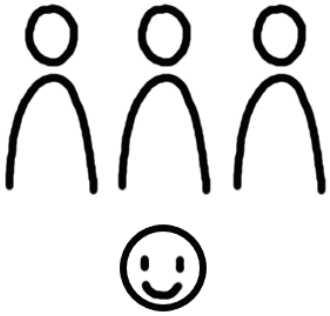


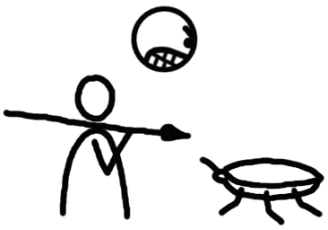
Nun, wir kannten diese Definition damals noch nicht. Bei uns galt einfach: «Kein Bug überlebt den Sprint.» Um das zu erreichen, mussten wir aber zuerst runter auf null Bugs. Wieder hatten wir Glück und die Projektleitung gewährte einen Aufschub, sodass wir uns vollständig auf die Bugs konzentrieren konnten. Es dauerte mehrere Sprints, aber am Ende hatten wir unser Ziel erreicht: keine bekannten Bugs im System.

DIE EFFEKTE VON ZERO BUGS

Von nun an ging es darum, diese Zahl zu halten. Wurden während eines Sprints neue Bugs entdeckt, mussten sie bei nächster Gelegenheit behoben werden. Verschiedene Teams packten dies unterschiedlich an. Ein Team hatte die Regel, dass man zwischen zwei Storys zuerst einen Bug beheben musste. Ein anderes Team definierte pro Sprint eine Person, die sich vor allem auf Bugs konzentrierte. Ein weiteres Team koordinierte das Bugfixing einfach am Daily Scrum.

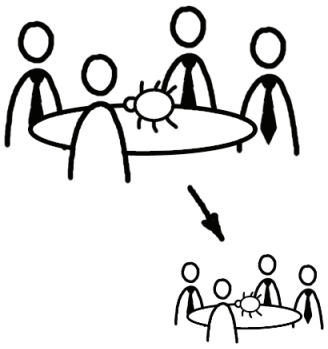
Unabhängig davon, wie die Zero Bug Policy umgesetzt wurde, geschahen in allen Teams kurz nach der Einführung erstaunliche Dinge. Das Gefühl der Hoffnungslosigkeit verschwand von einem Tag zum andern. Es war, als wäre allen eine schwere Last vom Herzen gefallen.

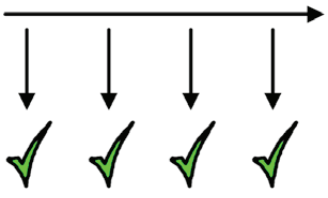




Vor der Zero Bug Policy war jeder neu entdeckte Bug nur einer von vielen. Bugfixing war ein notwendiges Übel, das einen von der spannenderen Arbeit abhielt. Mit der Zero Bug Policy verletzte ein neuer Bug den Stolz des Teams. Es entstand ein regelrechter Blutrausch. Der Scrum Master musste Teammitglieder sogar davon abhalten, alles stehen und liegen zu lassen, wenn jemand einen neuen Bug entdeckte.

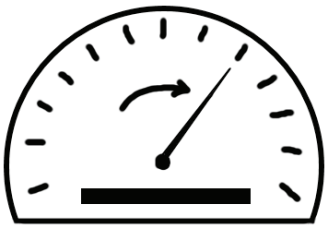
Das Bug Management verschwand nicht, wurde aber viel einfacher. Ein Beispiel dafür sind Bug-Duplikate. Bei mehreren hundert Bugs ist es ziemlich aufwändig festzustellen, ob ein Bug bereits einmal erfasst wurde. Mit der Zero Bug Policy ist es simpel. Im Idealfall hat man null Bugs. Da erübrigt sich die Suche nach Duplikaten. Auch mit beispielsweise fünf bekannten Bugs ist die Aufgabe immer noch trivial.





Sogar auf das Ergebnis eines Sprints hatte die Zero Bug Policy einen positiven Effekt. Eines der Ziele jeden Sprints liegt darin, das Produkt möglichst nahe an einem auslieferbaren Zustand zu haben (potentially shippable). Mit Hunderten bekannter Bugs war an eine Auslieferung natürlich nicht zu denken. Dank der Zero Bug Policy kamen wir dem Ziel nun einen grossen Schritt näher.

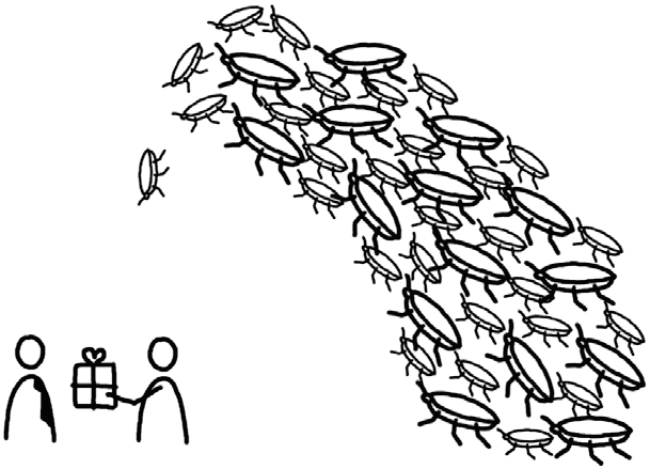
Darüber hinaus veränderte sich auch die Zusammenarbeit mit der nachgeschalteten Testgruppe. Früher fühlte sich eine Auslieferung eines Sprints so an: «Hier habt ihr das neue Produkt. Findet die zwanzig neuen Bugs, die wir zusätzlich zu den hundert bekannten seit dem letzten Sprint eingebaut haben.» Mit der Zero Bug Policy änderte sich unsere Einstellung: «Hier habt ihr das neue Produkt. Es hat keine bekannten Bugs. Beweist, dass wir falsch liegen.»



Und schliesslich gab es einen Effekt der Zero Bug Policy, der den beträchtlichen initialen Aufwand und alle Risiken rechtfertigte, die mit solch einer radikalen Methode verbunden sind. Unsere Velocity stieg mit Einführung der Zero Bug Policy um mindestens 30 Prozent an. Ich kann nicht mit Sicherheit sagen, was den Unterschied ausmachte. War es die gesteigerte Motivation aller Beteiligten? War es der Umstand, dass wir weniger Zeit mit Workarounds verbrachten? War es das reduzierte Bug Management? War es die generell höhere Qualität des Produkts? War es alles zusammen? Oder war es etwas ganz anderes? Eigentlich spielt es keine Rolle. Entscheidend war nur, dass wir durch die Zero Bug Policy die Arbeit, die nicht zum Wert des Produkts beitrug (Waste), beträchtlich reduzierten.

IST ES TOT?

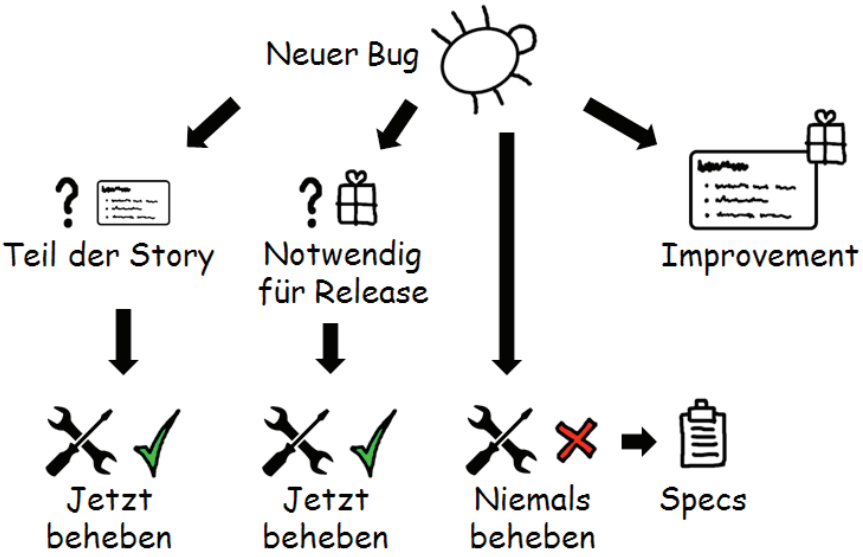
War das unser Happy End? Leider noch nicht ganz. Wie in so vielen Geschichten von Hollywood war das Monster noch nicht wirklich besiegt und griff gerade dann noch ein letztes Mal an, als wir uns alle schon in Sicherheit wiegten.



Version 1.0 war fertiggestellt. Dies war die erste Version, die an einen echten Kunden ausgeliefert wurde. Als die Teams damit begannen, an der nächsten Produktversion zu arbeiten, geschah etwas Unerwartetes. Die Bug-Zahl stieg mit einem Mal bei allen Teams sprunghaft auf eine hohe zweistellige Zahl an. Was war passiert?

Wir hatten einen wichtigen Punkt der Zero Bug Policy nicht umgesetzt. Statt zu entscheiden, ob ein Bug behoben werden soll oder nicht, hatte das Bug-Management-Team einfach alle Bugs, die nicht in Version 1.0 behoben werden mussten, auf eine zukünftige Version verschoben.

War das wirklich falsch? Muss man für die Zero Bug Policy wirklich immer die harte Entscheidung treffen, ob man einen Bug sofort oder nie behebt? Auf der Suche nach einer Antwort fanden wir eine umfangreichere Definition der Zero Bug Policy.



Als Erstes fragt man sich: Gehört der gefundene Bug zu einer Story des aktuellen Sprints? Wenn ja, dann ist die Story nicht abgeschlossen und der Bug muss sofort behoben werden. Dies scheint offensichtlich. Aber wir hatten tatsächlich die Situation, dass Teams Storys mit offenen Akzeptanzkriterien abschlossen, indem sie einfach für die offenen Punkte Bugs erfassten.

Die zweite Frage kannten wir bereits: Muss der Bug behoben werden, bevor das Produkt ausgeliefert wird? Wenn ja, dann wird der Bug sofort behoben.

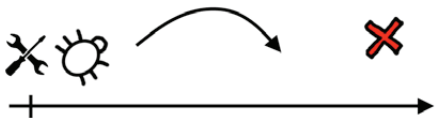
Ist die Reparatur eines Bugs nicht zwingend, bietet aber einen grossen Mehrwert, wird aus dem Bug eine Optimierung (Improvement). Diese übernimmt man als Story in die Planung der nächsten Produktversion. Was initial als Bug galt, wird aus dem Bug-Tracking entfernt und ist für die aktuelle Version ein erwartetes Verhalten.

Und schliesslich kann man sich auch entscheiden, einen Bug wirklich nie zu beheben. Dies ist typischerweise bei Bugs der Fall, die sehr selten auftreten, kein Risiko verursachen, deren Behebung sehr viel kostet und sehr wenig Mehrwert bringt.

Unabhängig davon, ob man einen Bug nie beheben will oder als Optimierung für die nächste Produktversion plant, wird der Bug zu einem erwarteten Verhalten. Eine allfällige Spezifikation sollte dies reflektieren. Werden die Spezifikationen aus automatischen Akzeptanztests generiert, ist dies nicht besonders schwierig. Man fügt einfach für das erwartete Verhalten einen weiteren Test hinzu.

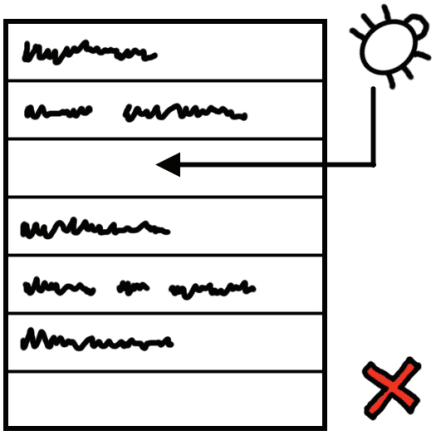
WIESO NICHT ...?

«Wieso muss ich einen neuen Bug sofort beheben? Wir haben momentan Wichtigeres zu tun. Kann ich nicht einfach meine Storys fertigmachen und den Bug auf den nächsten Sprint verschieben?» Solche Aussagen waren nicht ungewöhnlich während unserer ersten Schritte mit der Zero Bug Policy.

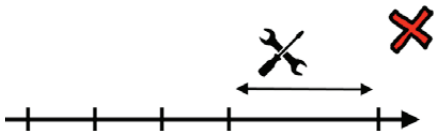


Es hat sich aber gezeigt, dass das Verschieben von Bugs kaum Vorteile brachte. Im nächsten Sprint hatte man wieder wichtigere Arbeit zu erledigen und es kamen neue Bugs hinzu. «Ich mache es später, wenn ich mehr Zeit habe» hat für uns nie wirklich funktioniert.

Vor der Zero Bug Policy hatten wir auch eine Weile versucht, Bugs ins Backlog einzufügen. Bugs waren aber schwer einschätzbar und unser Backlog begann stark zu fluktuieren. Ausserdem waren unsere Product Owner nicht an Bugs interessiert. Falls also etwas aus einem übervollen Sprint entfernt werden musste, waren es immer zuerst Bugs.



Schliesslich gab es eine Weile den Plan, die Bugs am Ende der Entwicklung in der Stabilisierungsphase zu beheben. Dann sind aber die meisten Bugs schon sehr alt, was zusätzlichen Aufwand verursacht. Ausserdem ist das auch für den Release-Plan ein beträchtliches Risiko. Angenommen, die Wahrscheinlichkeit, dass mit der Behebung eines Bugs ein neuer Bug eingeführt wird, liegt bei 30 Prozent. Bei 300 bereits bekannten Bugs, die in der Stabilisierungsphase behoben werden, erhalten wir also etwa 100 neue Bugs. Da ist es ziemlich wahrscheinlich, dass einer der neuen Bugs ein Show-Stopper ist, der Einfluss auf den Release-Plan hat.



Mit der Zero Bug Policy gibt es keine bekannten Bugs zu Beginn der Stabilisierungsphase und damit auch kein Risiko für zusätzliche Show-Stopper durch spätes Bugfixing.

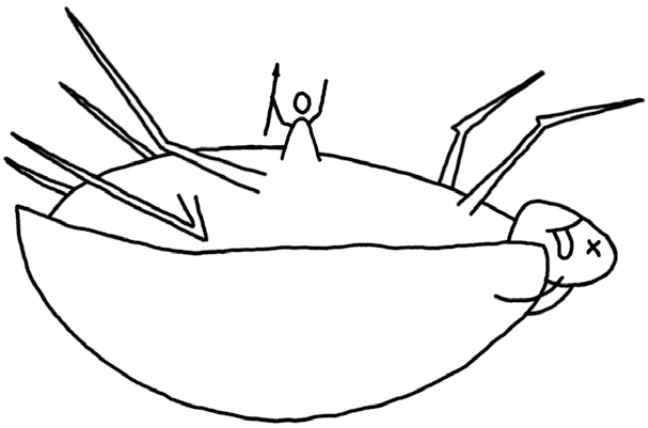
Trotzdem werden natürlich während der Stabilisierungsphase immer noch bisher unentdeckte Bugs gefunden und repariert. Show-Stopper können also immer noch überraschend auftreten. Sie sind aber viel weniger wahrscheinlich, da zu diesem Zeitpunkt dank der Zero Bug Policy insgesamt viel weniger Reparaturen und damit Code-Änderungen stattfinden.

HAPPY END?

Uns gelang es damals, die Zero Bug Policy nach dem ersten Rückschlag in einer verbesserten Version neu zu starten, und wir haben es nicht bereut. Natürlich war die Zero Bug Policy in unserem Fall zu einem gewissen Grad Symptombekämpfung. Eine Ursache unserer Probleme war sicherlich, dass die Anzahl der Teams in kurzer Zeit drastisch erhöht wurde, um einen Termin zu halten. Dass die Entwicklung und das Testen in verschiedenen Abteilungen stattfanden, war eine andere.

Trotzdem bin ich begeistert von der Zero Bug Policy und was sie bei uns bewirkt hat. Sie ist einfach, wirksam und am Anfang einer Entwicklung auch sehr kostengünstig einzuführen. Hat man bereits Hunderte von Bugs, ist der Wechsel etwas mühsamer, aber nicht unmöglich, wie es sich an unserem Beispiel gezeigt hat.

Es mag schwierig aussehen …



… aber es lohnt sich!

AUTOR

Adrian Krummenacher, Elektronik-Ingenieur HTL Fachrichtung Informatik, arbeitet seit 2000 für die bbv Software Services AG. Während dieser Zeit unterstützte er Kunden bei der Entwicklung von Softwaresystemen in den Bereichen Robotik, Bilderkennung, Abrechnungswesen und medizinische Diagnostik. Adrian ist ein überzeugter Anwender der agilen Produktentwicklung und begleitet seit über 10 Jahren agile Teams als Scrum Master.

REFERENZEN

[1] Davison, J. (Producer), Marshal A. (Producer), Verhoeven, P. (1997) Starship Troopers [Motion Picture]. United States: Columbia TriStar Motion, Buena Vista International

bbv
MAKING VISIONS WORK.

Über bbv

bbv Software Services ist ein Schweizer Software- und Beratungsunternehmen, das Kunden bei der Realisierung ihrer Visionen und Projekte unterstützt. Wir entwickeln individuelle Softwarelösungen und begleiten Kunden mit fundierter Beratung, erstklassigem Software Engineering und langjähriger Branchenerfahrung auf dem Weg zur erfolgreichen Lösung.

MAKING VISIONS WORK.