Antipattern of the Month: Unresolved Proxy

by Ian Mitchell MVB · May. 03, 18 · Agile Zone · Opinion

Just as a Scrum Master is considered to be a "Servant Leader" to his or her team, so a Product Owner might be described as a "Value Maximizer" representing the interests of stakeholders. These are the euphemisms which perhaps best describe the accountabilities of each respective job. Both roles are, of course, essential to Scrum, and yet they do not scale in quite the same way. For example, many development teams, each with its own Scrum Master to help it, could potentially draw work from a single Product Backlog. That backlog must, however, be curated by exactly one Product Owner. Only the PO has this one-to-one relationship with the product being developed.

In Scrum, empiricism is achieved by releasing increments of completed work early and often. Product ownership can be seen as the associated art and science of maximizing value for the benefit of stakeholders. Who counts as a stakeholder, though?

Well, a stakeholder is really anyone who is significantly impacted by the product. The most obvious stakeholders might be those who are funding the initiative and are able to articulate their interest to a PO. Other stakeholders can be much further removed. For example, the dependents of people with savings accounts can be seen as stakeholders in the bank's IT infrastructure, even though those dependents may be quite unaware of any technical systems, or even of the accounts existing at all. If any infrastructure was to fail, they would nevertheless be affected. A good PO exercises a moral duty to represent and balance all stakeholder interests in a product, and to provide a clear authoritative voice to the Development Team concerning the necessary requirements. The role cannot be filled by a committee.

Finding a Product Owner who has the above competencies is a huge challenge for many organizations. Not only must someone have the skills, they must have the time to spend on an emergent product. Very often the best-qualified person can be "high up" in the organization, and is seen as "too busy" to make the necessary commitment. That's when a "proxy" Product Owner is often resorted to. The idea is that they should provide effective representation in the field, and only involve the genuine PO when things get out of hand or certain defined tolerances are exceeded. In practice, this can be seen as a special case of *Management by Exception*.

Any proxy must be respected as having executive authority regarding value, so as not to be undermined. This includes authority over the articulation and ordering of work on a Product Backlog and how it is represented to the Development Team. The proxy must be a genuine and competent representative of the "real" PO, and recognized as being fully able to take decisive action and to provide information in a timely way.

Unfortunately, though, a proxying model can be resorted to as a salve when genuine product ownership is weak. Stakeholders might expect a certain product capability to be available, but none may necessarily wish to own it. This can be the case with middleware for example. Several proxies might then be used, each of whom will represent certain capabilities on behalf of a notional though absent Product Owner. Great discipline is needed when a single clear proxy is unrecognized, since all must then agree to establish compensatory protocols through which they collaborate beyond their narrow interests.

Remember that a strong case can be made against having a "proxy" PO of any type. After all, if a product does not have a true owner, and there is no clear authority for its value to be maximized, is it really worth developing in the first place?

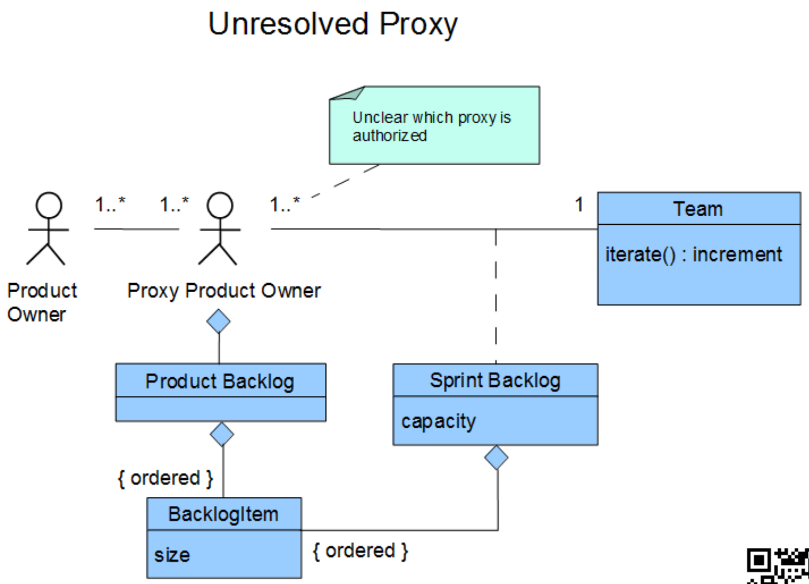
Unresolved Proxy is Antipattern of the Month at agilepatterns.org

Unresolved Proxy

Intent: Conceal weak Product Ownership by distributing it

Proverbs: That which is owned by everyone ends up being owned by no-one

Motivation: Multiple stakeholders can have an interest in a product and each may only be able to articulate the requirements in their area. Consequently, a senior product owner may be tempted to delegate ownership to multiple proxies.

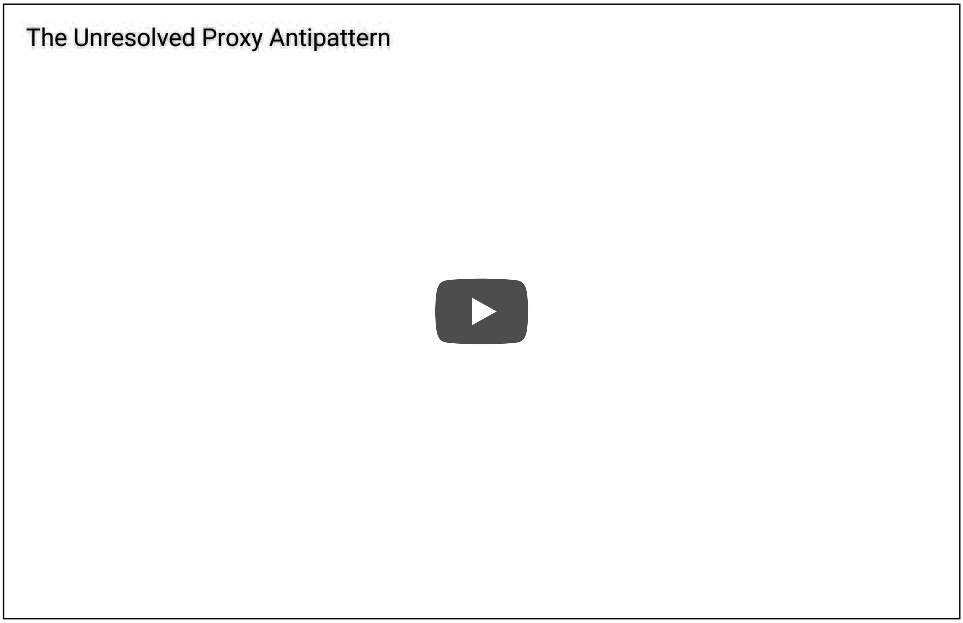


Structure: A development team tries to negotiate a Sprint Backlog with a Product Owner. However, the PO is represented by multiple proxies and it is unclear which one has authority over the Product Backlog and the backlog items which are to be negotiated.

Applicability: Unresolved proxies can be found where Product Owners do not exercise authority or control over the Product Backlog or how it is represented. They are especially common in projects where multiple stakeholders consume a service, such as that which may be provided by shared components or middleware.

Consequences: Sprint Planning will be compromised as it is unclear which proxy represents the Product Owner and has authority to negotiate a Sprint Backlog. Increments are likely to be of low quality if the team do not know who to consult with regarding the approval of work. The Definition of Done may be unclear.

Implementation: Scrum does not provide for proxy product ownership, although it is widely used and misapplied in a Scrum context. DSDM provides for several business roles which can reduce to unresolved ownership of a backlog (Prioritized Requirements List) unless discipline is maintained. XP mandates that the team liaise with a clear customer role and hence proxying is effectively forbidden.



Learn more about the myths about Scrum and DevOps. Download the whitepaper now brought to you in partnership with Scrum.org.

Like This Article? Read More From DZone

Scrum: 19 Sprint Planning Anti-Patterns

Lipstick Agile: 13 Signs You Probably Need a New Job

Pattern of the Month: Proxy Product Ownership

Free DZone Refcard
Software Configuration Management Patterns

Topics: AGILE PATTERNS , PRODUCT OWNERSHIP , PRODUCT OWNER , PRODUCT OWNER PROXY , AGILE , AGILE ANTI PATTERNS

Best Practices for Reducing Down Time on the Way to Production

by Serhat Can MVB · May 11, 18 · Agile Zone · Opinion

Discover how TDM Is Essential To Achieving Quality At Speed For Agile, DevOps, And Continuous Delivery. Brought to you in partnership with CA Technologies.

Incident management is often associated with production incidents that require immediate action. But, not all incidents are alike. Incidents can occur when integrating your code into master, releasing your software, changing a configuration item, or receiving a malicious email.

At OpsGenie, an on-call and incident management solution, we aim to reduce your alert fatigue and help you manage all kinds of incidents in the right way. Here are some best practices that we have learned along the way that will help your teams develop, release, and run software with confidence.

Monitor: Stay Vigilant on the Way to Production and Alert Teams of Issues

In DevOps, it is important to keep feedback loops small and active so that teams can release and fix problems faster. Continuous integration and delivery tools can alert teams of the issues that may break the release pipeline and block the way to production. It is dangerous to not be able to ship code because a bug may arise or a revert might be necessary. That is why it becomes critical to notify the on-call responder and keep everyone involved to maintain a healthy release pipeline.

Orchestrate: Gather and Automate Your Alerts from Different Tools

The number of tools used by organizations to ease their DevOps process increases daily. Depending on the level of abstraction, monitoring needs may require network, application performance, SLA, API monitoring, and more.

Development teams that embrace agile practices leverage tools like Jira or Zendesk for ticketing, and Slack or MS Teams for messaging. Considering how integrated those tools can be and the number of them that keep relevant information that concerns other stakeholders, it is hard but very critical to keep everything organized. OpsGenie updates other tools in your stack by opening tickets or closing alerts automatically. Tight integrations with chat tools like Slack allow teams to take full advantage of ChatOps by bringing day-to-day operations into shared chat channels.

Prioritize: Don't Page for Low Priority Incidents

Not all incidents are the same. Some might have catastrophic effects on multiple services, while others require only a check. While collecting alerts from testing, and monitoring tools like AlertSite, teams can add tags or assign different priorities to alerts. Those priorities can then be used to route your alerts to different escalations.

For example, a low priority incident can use an escalation with one developer in it, and even suppress notification for an hour, while a high priority incident can notify the on-call immediately and if on-call does not acknowledge the alert, then the whole team might be paged. This escalation policy can go all the way up to CTO depending on the priority.

Cluster: Group-Related Alerts

Incidents rarely create just 1-2 alerts. Think about a case where a core service in your microservices break and multiple dependent services are affected, or when the build is broken and dependent teams can't integrate their code into production. In real life, incidents are complex.

In complex incidents, the on-call responder might receive hundreds of alerts from various levels of testing and monitoring solutions. It is important to have a clear view of related alerts and separate unrelated ones. The biggest problem is to be so drowned by alerts that you cannot find important pieces of information that will help your team resolve the issue. Or worse, to miss other key alerts in the mess.

This is why OpsGenie introduced its new incident response orchestration solution. Users can group alerts automatically or manually and notify subscribers or predefined stakeholders about the issue. The value is in reducing the alert fatigue with smart grouping capabilities and managing the dependency between affected parties.

Communicate: Keep Stakeholders in the Loop

As we talked about earlier, complex incidents involve multiple teams. Those teams may have a part in the incident or just use their services and might need to take preventative actions to protect themselves. It is important to keep them up to date.

However, this is easier said than done. Service dependencies and responsible teams should be defined and alerts should be routed to those teams or individual stakeholders. Status pages and health dashboards help teams to subscribe to updates and gather quick feedback on the status of the issue without bothering the teams that are already busy solving the problem.

How do AlertSite and OpsGenie help together?

AlertSite users can integrate Smartbear's with OpsGenie and use OpsGenie's powerful incident response orchestration and alerting capabilities to improve uptime for REST and SOAP APIs.

Learn more at OpsGenie's AlertSite integration page and signup for a free trial to see how it can help you keep your service up and running!

See how three solutions work together to help your teams have the tools they need

See how three solutions work together to help your teams have the tools they need to deliver quality software quickly. Brought to you in partnership with CA Technologies.

Like This Article? Read More From DZone

Make Alerting Apps Work for You

Incident Management and Continuous Integration, a Partnership for Success

All Day DevOps 2017: The Unrealized Role of Monitoring and Alerting

Free DZone Refcard
Software Configuration Management Patterns

Topics: AGILE, DEVOPS, INTEGRATION, ALERTS, PERFORMANCE, ERRORS, INCIDENT MANAGEMENT