Context Ticket inv:   
self.timestampResolved > self.timestampCreated  
*-- De datum voor het oplossen van het ticket moet na de aanmaakdatum van het ticket liggen*

Context Ticket inv:   
self timestampProjectedViolation > self.timestampCreated  
*-- De datum voor overschrijden van de SLA-termijnen, moet na de aanmaakdatum van het ticket liggen*

Context Ticket::status: enumStatus  
init: self.status = enumStatus.Open  
*-- Ieder ticket begint in de status “Open”*

Context Ticket inv:  
self.timestampResolved not null implies self.status = enumStatus.Resolved or enumStatus.Rejected or enumStatus.Failed  
*-- Indien een tijdstip voor afsluiten van het ticket is ingesteld, moet het ticket status “Resolved”, “Rejected” of “Failed” hebben*

Context Ticket inv:  
if self.impact = enumImpact.High and self.Urgency = enumUrgency.High  
 then self.Priority = enumPriority.Critical  
if (self.impact = enumImpact.High and self.Urgency = enumUrgency.Medium) or (self.impact = enumImpact.Medium and self.Urgency = enumUrgency.High)  
 then self.Priority = enumPriority.High  
if (self.impact = enumImpact.High and self.Urgency = enumUrgency.Low) or (self.impact = enumImpact.Medium and self.Urgency = enumUrgency.Medium) or (self.impact = enumImpact.Low and self.Urgency = enumUrgency.High)  
 then self.Priority = enumPriority.Medium  
if (self.impact = enumImpact.Medium and self.Urgency = enumUrgency.Low) or (self.impact = enumImpact.Low and self.Urgency = enumUrgency.Medium)  
 then self.Priority = enumPriority.Low  
if self.impact = enumImpact.Low and self.Urgency = enumUrgency.Low  
 then self.Priority = enumPriority.Planning  
*-- Implementatie van de priority-matrix uit de opgave*

Context Ticket inv:  
let totalTime: double = Ticket.actions->sum(a | a.timeTaken)  
self.timeSpent = totalTime  
*-- De gespendeerde tijd voor een ticket is de som van de tijd die nodig was voor alle uitgevoerde acties*

Context Ticket inv:  
self.Children->Count() > 0 implies self.Children->forAll(c | c.parent = self)  
*-- Als het ticket child-tickets heeft, moet de parent van ieder child ticket het ticket zelf zijn*

Context Ticket inv:  
self.status = enumStatus.Resolved or enumStatus.Rejected or enumStatus.Failed implies   
self.children->forAll(c | c.status = emumStatus.Resolved or enumStatus.Rejected or enumStatus.Failed)  
*-- Als een ticket children heft (en dus een parent-ticket is), kan het enkel worden afgesloten als alle children ook zijn afgesloten*

Context Ticket::assignToHandler(): void  
pre: self.handledBy->isEmpty()  
post: self.handledBy->notEmpty()  
*-- Een ticket toewijzen aan een behandelaar kan enkel als het nog niet is toegewezen*

Context Ticket::changeStatus(): void  
pre: self.status = not (enumStatus.Resolved or enumStatus.Rejected or enumStatus.Failed)  
*-- Een ticket dat een van de afgesloten statussen heeft, kan niet meer van status worden veranderd*

Context Ticket::appendChildTicket(child: Ticket):void  
pre: child.Parent->isEmpty()  
post: self.children->includes(child) and child.parent->includes(self)  
*-- Indien een child ticket wordt toegekend aan een ticket, mag het toegewezen ticket nog niet tot een andere parent behoren. Na de actie zal het parent-ticket naar het child-ticket verwijzen en zal het child-ticket tot het parent-ticket behoren*

Contect Ticket::appendToParentTicket(parent: Ticket): void  
pre: self.Parent->isEmpty()  
post: self.parent->includes(parent) and parent.children->includes(self)  
*-- Indien een ticket aan een parent wordt toegekend, mag het zelf nog niet aan een andere parent zijn toegekend. Na de actie zal het ticket naar het parent-ticket verwijzen en zal de parent naar het child-ticket verwijzen*

Context Ticket::relateToAsset(asset: Asset): void  
pre: self.relatedAssets->excludes(asset)  
post: self.relatedAssets->includes(asset)  
*-- Indien een asset aan een ticket wordt gekoppeld, mag het nog niet aan dat ticket zijn gekoppeld.*